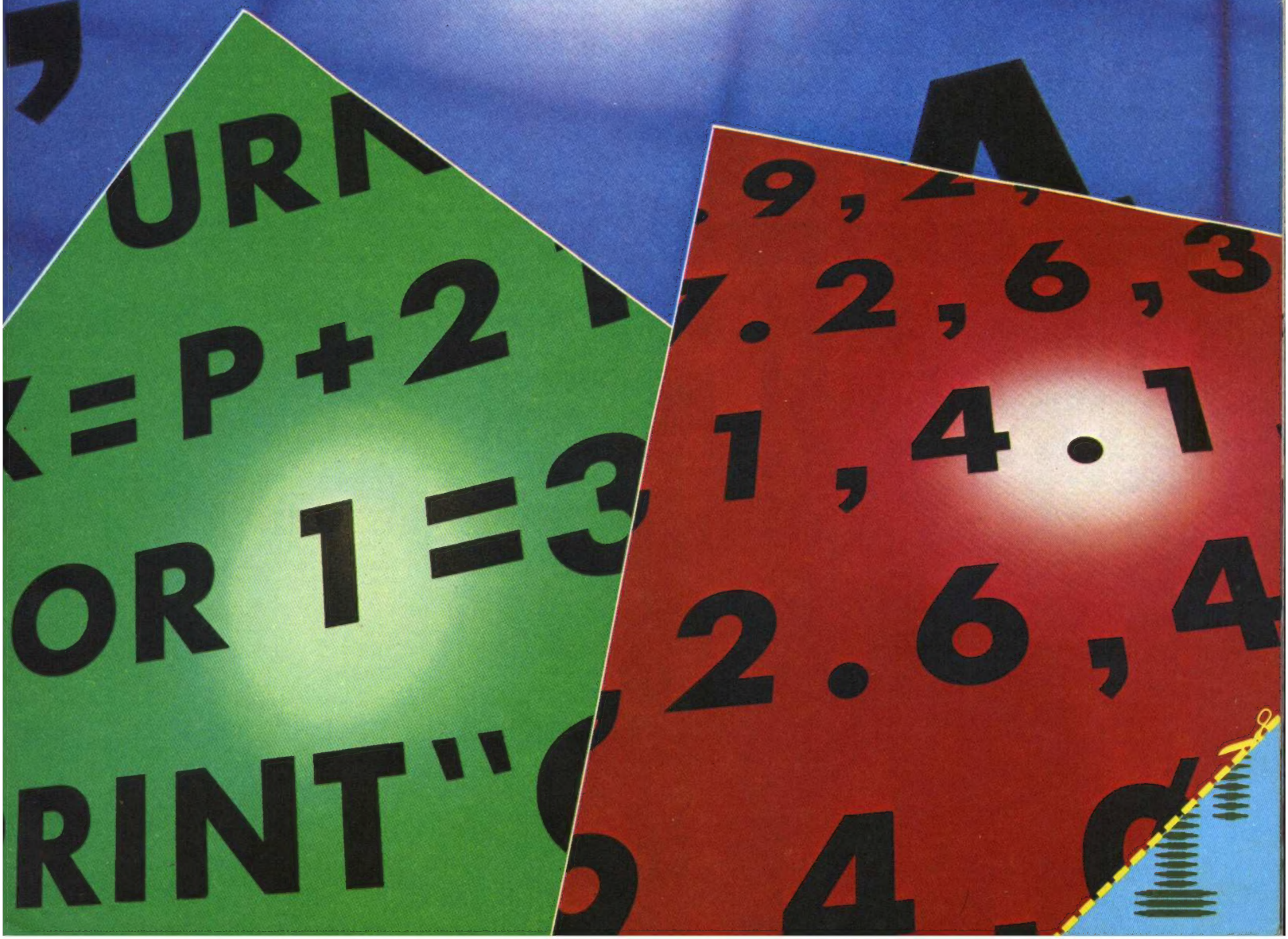
CURSO PRÁTICO 16 DE PROGRAMAÇÃO DE COMPUTADORES

TK!83

PROGRAMAÇÃO BASIC - PROGRAMAÇÃO DE JOGOS - CÓDIGO DE MÁQUINA

NOVA CULTURAL

Cz$ 20,00



INPUT

Vol. 2 N.º 16

NESTE NÚMERO

PROGRAMAÇÃO BASIC

RELAÇÕES MUITO LÓGICAS

O uso de operadores relacionais e lógicos. O significado das tabelas-verdade..... 301

PROGRAMAÇÃO DE JOGOS

OS OBJETOS DA AVENTURA

Linhas **DATA** para a lista de objetos. Colocação dos objetos nas posições corretas. Como pegar e largar objetos. Como mostrar ao jogador a lista dos objetos que carrega..... 306

PROGRAMAÇÃO BASIC

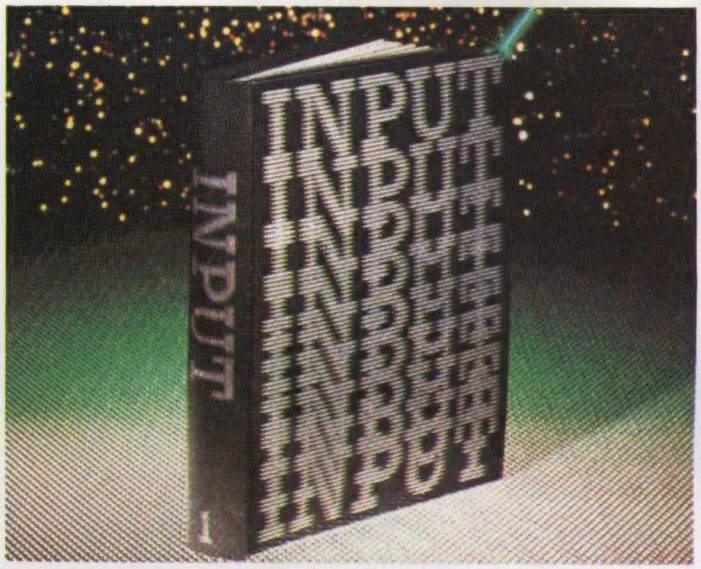
COMO EVITAR E DETECTAR ERROS

Os erros mais comuns. Localização dos erros e "depuração" do programa. Um teste para sua habilidade em detectar erros 311

CÓDIGO DE MÁQUINA

FIGURAS MÓVEIS

Como movimentar figuras no Apple e no ZX-81. Moto e submarino no Apple. Monstro no ZX-81 ... 316



PLANO DA OBRA

"INPUT" é uma obra editada em fascículos semanais, e cada conjunto de 15 fascículos compõe um volume. A capa para encadernação de cada volume estará à venda oportunamente.

COMPLETE SUA COLEÇÃO

Exemplares atrasados, até seis meses após o encerramento da coleção, poderão ser comprados, a preços atualizados, da seguinte forma: **1. Pessoalmente** — por meio de seu jornaleiro ou dirigindo-se ao distribuidor local, cujo endereço poderá ser facilmente conseguido junto a qualquer jornaleiro de sua cidade. Em São Paulo os endereços são: Rua Brigadeiro Tobias, 773 (Centro); Av. Industrial, 117 (Santo André); e, no Rio de Janeiro: Rua da Passagem, 93 (Botafogo). **2. Por carta** — Poderão ser solicitados exemplares atrasados também por carta, que deve ser enviada para DINAP — Distribuidor Nacional de Publicações — Números Atrasados — Estrada Velha de Osasco, 132 (Jardim Tereza) — CEP 06000 — Osasco — São Paulo. **3. Por telex** — Utilize o nº (011) 33670 ABSA. Em **Portugal**, os pedidos devem ser feitos à Distribuidora Jardim de Publicações Ltd. — Qta. Pau Varais, Azinhaga de Fetais — 2685, Camarate — Lisboa; Tel. 257-2542 — Apartado 57 — Telex 43 069 JARLIS P.
Não envie pagamento antecipado. O atendimento será feito pelo reembolso postal e o pagamento, incluindo as despesas postais, deverá ser efetuado ao se retirar a encomenda na Agência do Correio. **Atenção**: Após seis meses do encerramento da coleção, os pedidos serão atendidos, dependendo da disponibilidade de estoque. **Obs.:** Quando pedir livros, mencione sempre o título e/ou o autor da obra, além do número da edição.

COLABORE CONOSCO

Encaminhe seus comentários, críticas, sugestões ou reclamações ao Serviço de Atendimento ao Leitor — Caixa Postal 9442, São Paulo — SP.

Editor
VICTOR CIVITA

REDAÇÃO

Diretora Editorial: Iara Rodrigues

Editor chefe: Paulo de Almeida
Editor de texto: Cláudio A.V. Cavalcanti
Editor de Arte: Eduardo Barreto
Chefe de Arte: Carlos Luiz Batista
Assistentes de Arte: Ailton Oliveira Lopes, Dilvacy M. Santos, José Maria de Oliveira, Grace A. Arruda, Monica Lenardon Corradi
Secretária de Redação/Coordenadora: Stefania Crema
Secretários de Redação: Beatriz Hagström, José Benedito de Oliveira Damião, Maria de Lourdes Carvalho, Marisa Soares de Andrade, Mauro de Queiroz
Secretário Gráfico: Antonio José Filho

COLABORADORES
Consultor Editorial Responsável: Dr. Renato M.E. Sabbatini (Diretor do Núcleo de Informática Biomédica da Universidade Estadual de Campinas)
Execução Editorial: DATAQUEST Assessoria em Informática Ltda. Campinas, SP.
Tradução: Maria Fernanda Sabbatini
Adaptação, programação e redação: Abílio Pedro Neto, Aluísio J. Dornellas de Barros, Marcelo R. Pires Therezo, Raul Neder Porrelli
Coordenação geral: Rejane Felizatti Sabbatini
Editora de Texto: Ana Lúcia B. de Lucena
Assistente de Arte: Dagmar Bastos Sampaio
COMERCIAL
Diretor Comercial: Roberto Martins Silveira
Gerente Comercial: Flávio Ferrucio Maculan
Gerente de Circulação: Denise Maria Mozol

PRODUÇÃO
Gerente de Produção: João Stungis
Coordenador de Impressão: Atilio Roberto Bonon
Preparador de Texto/Coordenador: Eliel Silveira Cunha

Preparadores de Texto: Ana Maria Dilguerian, Antonio Francelino de Oliveira, Karina Ap. V. Grechi, Levon Yacubian, Maria Teresa Galluzzi, Paulo Felipe Mendrone
Revisor/Coordenador: José Maria de Assis
Revisoras: Conceição Aparecida Gabriel, Isabel Leite de Camargo, Ligia Aparecida Ricetto, Maria do Carmo Leme Monteiro, Maria Luiza Simões, Maria Teresa Martins Lopes.

© Marshall Cavendish Limited, 1984/85.
© Editora Nova Cultural Ltda., São Paulo, Brasil, 1986.
Edição organizada pela Editora Nova Cultural Ltda.
(Artigo 15 da Lei 5 988, de 14/12/1973).
Esta obra foi composta na AM Produções Gráficas Ltda. e impressa na Divisão Gráfica da Editora Abril S.A.

RELAÇÕES MUITO LÓGICAS

- OPERADORES RELACIONAIS
- MAIOR, MENOR, IGUAL
- OPERADORES LÓGICOS
- AND, OR E NOT
- TABELAS-VERDADE

Os computadores são capazes de executar milhões de operações lógicas em apenas um segundo. Mas a lógica é também parte fundamental de qualquer programa. Conheça seus operadores.

A programação BASIC utiliza três tipos de expressão: aritmética, de cadeia e lógica. As duas primeiras compõem a maior parte de qualquer programa. Porém, cabe às expressões lógicas a responsabilidade pelas decisões ao longo de um programa.

A função das expressões lógicas é, simplesmente, verificar se algo é "verdadeiro" ou "falso". As palavras e símbolos usados para isso em um programa são chamados de *operadores* (ou, ainda, *conectores*).

Nas expressões lógicas empregam-se dois tipos de operadores: os *operadores relacionais* (que incluem símbolos matemáticos como <, > e =) e os *operadores lógicos*: **AND**, **OR** e **NOT** (incluídos na lista de comandos de qualquer versão BASIC).

MAIOR, MENOR, IGUAL

Os operadores relacionais podem ser usados até no mais simples dos programas em BASIC. As comparações possíveis são:

A>B ..... **A** é maior que **B**
A<B ..... **A** é menor que **B**
A>=B .... **A** é maior ou igual a **B**
A<=B .... **A** é menor ou igual a **B**
A=B ..... **A** é igual a **B**
A<>B .... **A** não é igual a **B**

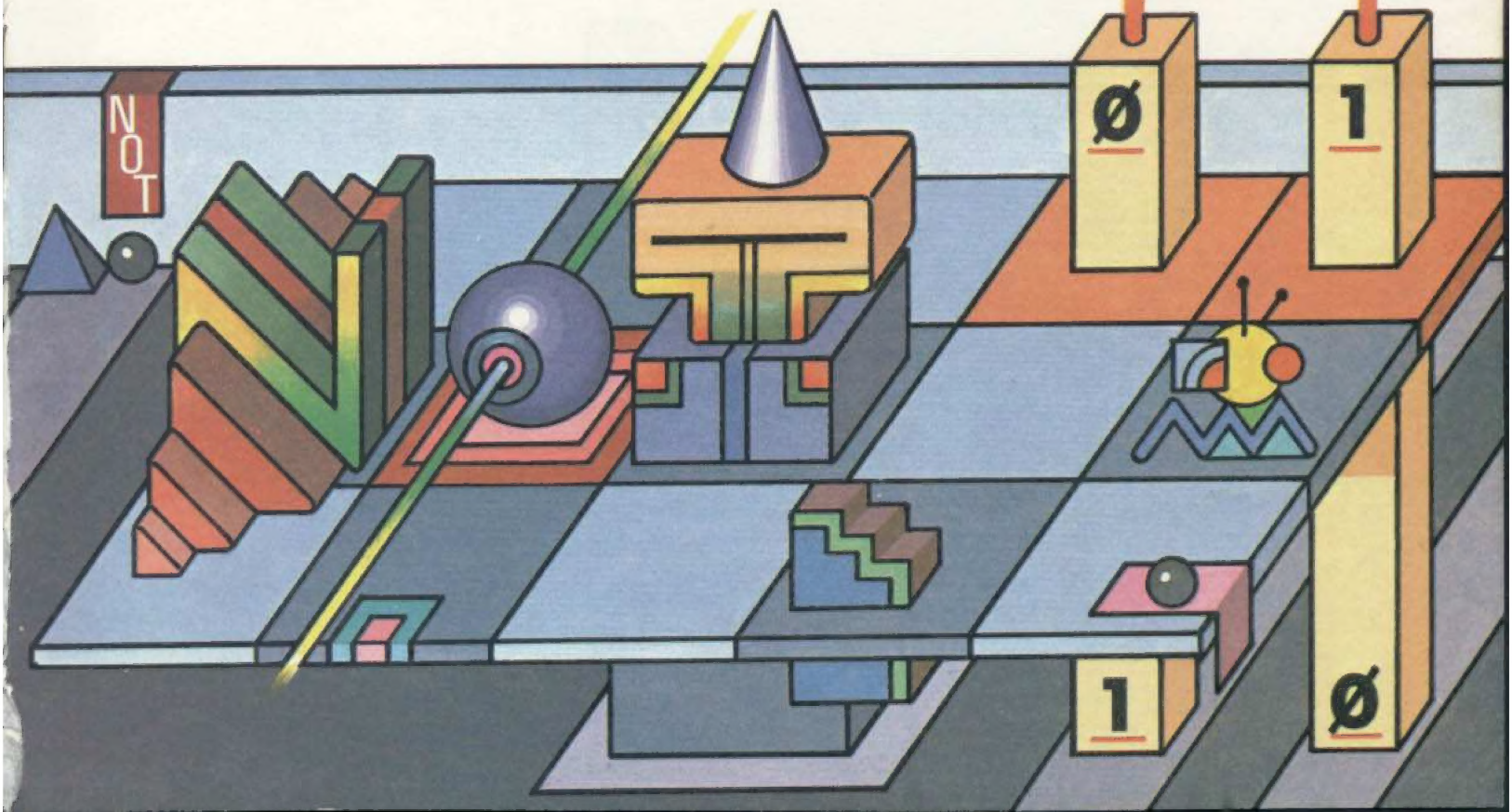
Você já deve ter visto esses operadores em vários programas de INPUT. Embora possam ser usadas em aritmética direta, é no uso conjunto com o **IF...THEN** que se torna mais evidente sua contribuição nos testes condicionais. Um exemplo comum:

```
IF A>B THAN PRINT" A É MAIOR
QUE B"
```

Neste caso, o valor de **A** deve ser maior que o de **B** para que o restante da linha seja executado. Sempre que o valor de **A** for menor que o de **B**, o programa saltará para a próxima linha. O exemplo torna evidente a possibilidade de um salto condicional: se um conjunto de valores satisfaz a condição, então o programa faz algo; se é um outro conjunto de valores que a satisfaz, então o programa faz outra coisa.

O uso dos operadores relacionais não se limita a valores numéricos. Eles também podem ser empregados para comparar cadeias. Contudo, isso requer certa cautela; caso contrário, obtêm-se resultados um tanto quanto estranhos. É importante lembrar, em primeiro lugar, que a comparação sempre pára no último caractere da cadeia mais curta. Ou seja, se uma cadeia contém onze caracteres e outra contém sete, somente os sete primeiros da cadeia mais longa serão comparados com os da cadeia mais curta.

Na realidade, a comparação é feita com um caractere de cada vez, da es-



querda para a direita — e aqui se encontra a explicação para o surgimento de resultados estranhos. Numa comparação numérica, a afirmação **5>10** é obviamente falsa mas, numa comparação entre cadeias, pode-se ter uma afirmação na forma **A$>B$**. Se **A$="5"** e **B$="10"**, o computador compara primeiro os caracteres à esquerda — **5** e **1**. Em seguida pára, pois para ele não há mais nada a ser comparado.

Assim, temos uma condição "verdadeiro" incorreta, pois **5** é maior que **1**, mas o computador ignorou o caractere que restou na cadeia mais longa. Esteja sempre atento a erros como este.

Nas cadeias, as letras do alfabeto seguem a seguinte ordem de comparação: **"A"<"B"<"C"<"D"**, e assim por diante. Mais uma vez, seja cauteloso, pois o computador não entende o significado dos caracteres. Na verdade, o que ele faz é comparar os códigos dos caracteres, o que explicaremos com mais detalhes num próximo artigo. Em conseqüência, os espaços e outros caracteres que não são letras tornam-se tão importantes quanto as próprias letras, pois cada um tem seu código. Esquecer-se disso resultaria em erro, por exemplo, num programa que coloca cadeias em ordem alfabética.

Se cada cadeia tem a mesma seqüência de caracteres, a mais longa será considerada a maior, numericamente. Mas note que a cadeia mais curta é tomada como a maior numa expressão como **"ABD">"ABCD"**, pois a comparação para quando encontra a primeira diferença — ou seja, quando os valores dos códigos de **"D"** e **"C"** são comparados. A prioridade, portanto, é parecida com aquela dos dicionários ou índices, onde "amor" vem antes de "amoroso" mas depois de "amargo".

VERDADEIRO OU FALSO

Depois de qualquer comparação, obtém-se um resultado — um número inteiro. Este é **0**, se a comparação for falsa, e **−1** ou **1** (dependendo da máquina), se for verdadeira.

Experimente inserir no seu microcomputador, em modo direto (imediato), a seguinte linha:

```
PRINT 6>5 , 5>7
```

A expressão da esquerda é verdadeira e a da direita, falsa. Você verá aparecer na tela, portanto, o resultado **1** (ou **−1**) e **0**.

Os resultados obtidos numa comparação podem ser usados em cálculos no decorrer do programa. Mas tome cuidado com a divisão: qualquer tentativa de dividir um número por **0** resultaria numa mensagem de erro.

OPERADORES LÓGICOS

AND, **OR** e **NOT** são operadores lógicos que auxiliam na tarefa de decisão do **IF...THEN** (página 41). Por exemplo: **IF** (se) isto é verdadeiro **AND** (e) aquilo é verdadeiro **THEN** (então) faça alguma coisa. Os outros dois operadores seriam usados da mesma maneira.

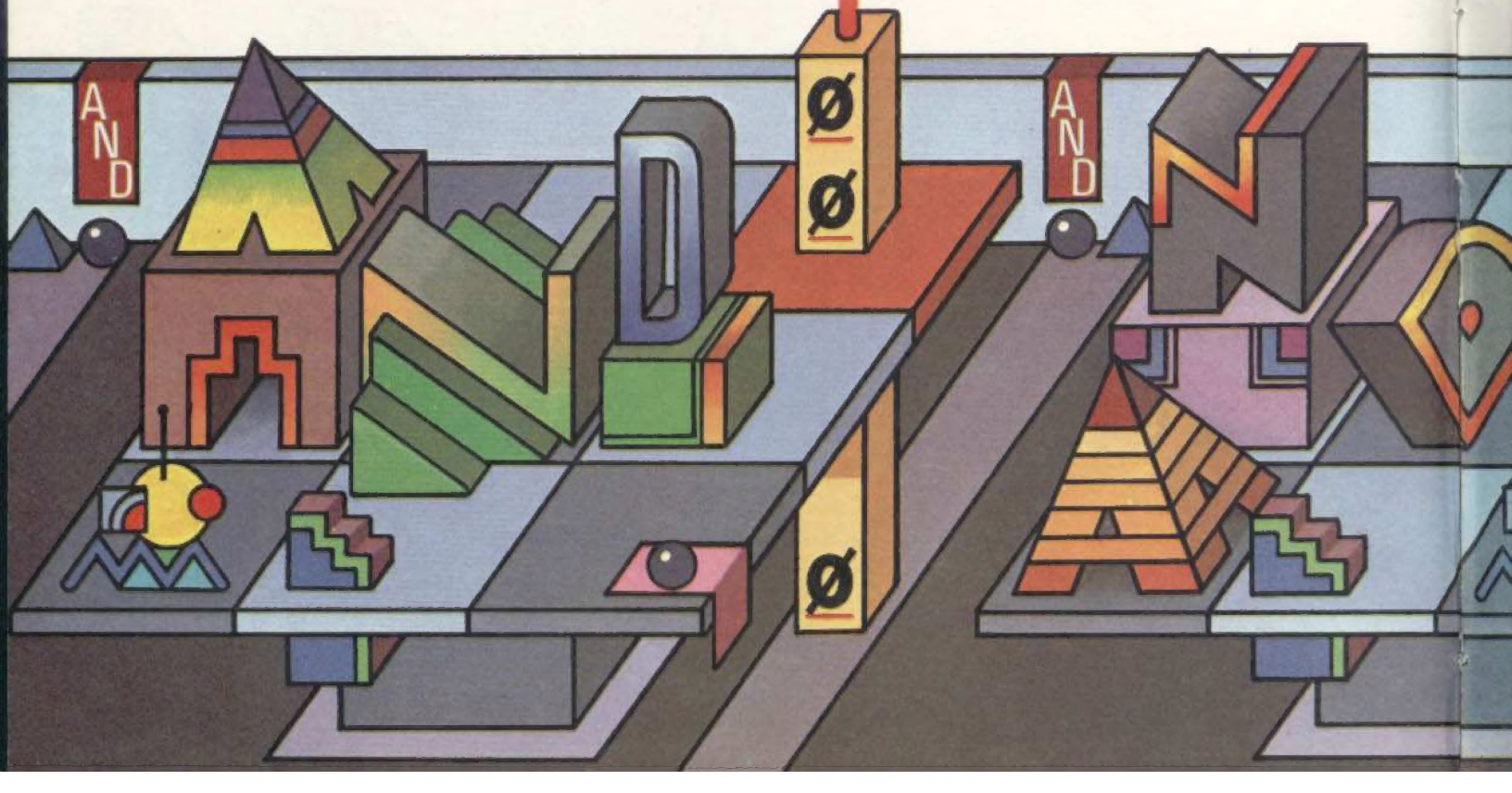
Estes operadores — às vezes chamados de *operadores booleanos* (inventados pelo matemático inglês George Boole) — podem ser empregados para comparar números ou cadeias, tanto no modo direto como no modo de programa, e obter um "valor verdade" para o que se considera uma condição de teste muito complexa. Eles oferecem um caminho mais curto para o cumprimento de uma tarefa que envolveria um amontoado de **IF...THEN**.

O USO DO AND

Simplificadamente, o operador **AND** pode ser considerado como tendo o significado de **E**. Numa expressão como:

```
IF V>0 AND V<100
PRINT" PERMITIDO "
```

...a mensagem aparece na tela somente se **V** for maior que **0** e menor que **100**.



Num programa, teríamos algo como:

```
990 OKAY=1 AND MES>5 AND MES<10
 AND ANO>1900 AND ANO<2001
```

```
990 OKAY=-1 AND MES>5 AND MES<1
0 AND ANO>1900 AND ANO<2001
```

Uma linha de programa como esta poderia ser empregada para vários fins — por exemplo, para verificar se uma certa data caiu no inverno, no século vinte.

Utiliza-se o **AND**, no caso, para comandar quatro testes. Todos devem ser verdadeiros para que a variável **OKAY** permaneça como verdadeira. Nesse teste de validade, primeiramente ajusta-se a variável **OKAY** como "verdadeira" (−1, ou 1 no Sinclair e Apple). Depois, a condição necessária é que **MES** esteja entre **6** e **9** e que **AND** esteja entre **1901** e **2000** (inclusive).

Verifiquemos mais de perto o que acontece: se as condições forem totalmente satisfeitas, todas as expressões produzirão um valor verdadeiro. Em conseqüência, a linha de programa ficaria assim:

```
990 OKAY= -1 AND -1 AND -1 AND
-1 AND -1
```

(nos micros da linha Sinclair e Apple seria 1, em vez de −1)

Se acrescentarmos **PRINT OKAY**, verificaremos que, de fato, o resultado é −1, ou seja, verdadeiro.

O USO DO OR

Embora o significado de **OR** (traduzindo, teríamos **OU**) seja diferente do de **AND**, ambos podem ser considerados semelhantes em relação à maneira como são usados.

Vejamos um exemplo simples, que emprega **OR** na comparação de valores de uma determinada variável.

```
IF V=8 OR V=10 PRINT" OKAY "
```

A mensagem aparecerá na tela se o valor de **V** for **8** ou **10**.

O emprego do **OR** é muito útil na verificação da validade dos dados introduzidos no computador via comando **INPUT**. Se temos, por exemplo, um programa que pergunta pela idade, com certeza haverá alguém que, só por curiosidade, responderá −10 ou 999.

Para contornar problemas como esse, usamos a seguinte alternativa:

```
10 INPUT A
20 IF A<1 OR A>120 THEN PRINT"
   SEJA SENSATO ": GOTO 10
```

O USO DO NOT

O terceiro operador lógico de uso bastante comum é o **NOT**. Ele difere um pouco dos outros, pois age somente na expressão numérica ou lógica que o segue — **AND** e **OR** comparam expressões nos dois lados, mas **NOT** trabalha sobre um único valor.

Veja um exemplo do uso do **NOT**.

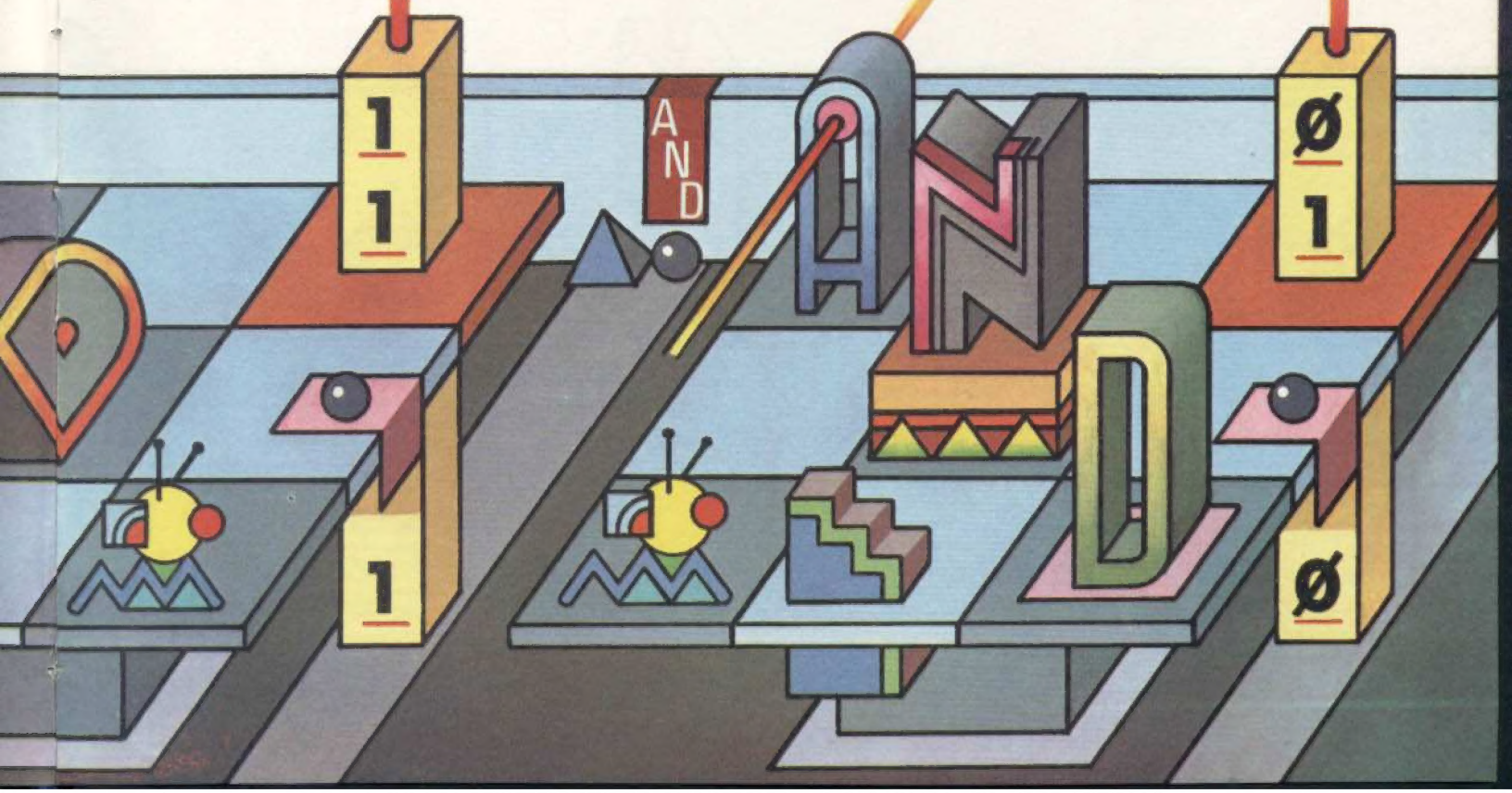
```
IF NOT (A>10) THEN 999
```

Se traduzíssemos isto para uma linguagem do dia-a-dia teríamos: "se o valor de **A** não é maior que **10**, então vá para a linha **999**".

A função lógica do **NOT** é converter para falsa uma condição verdadeira, ou vice-versa. Poderíamos utilizá-lo, por exemplo, como uma chave que inverte a condição falso/verdadeiro obtida no resultado de uma operação anterior.

TABELAS - VERDADE

Ouvimos falar de "tabela-verdade" sempre que o assunto é operadores lógicos. Bem, elas são muito simples. Sua



função consiste em fornecer uma representação visual do que acontece com os falso/verdadeiro quando usamos **AND** e **OR** sobre eles. **NOT** não precisa de tabela, já que produz sempre o inverso.

As tabelas podem assumir várias formas; uma típica para o **AND** é:

A	B	C	
-1	-1	-1	linha 1
-1	0	0	linha 2
0	-1	0	linha 3
0	0	0	linha 4

Para desvendar o significado da tabela, basta interpretá-la linha por linha. Linha 1: "Se A é verdadeiro e B é verdadeiro, então C também é verdadeiro". Linha 2: "Se A é verdadeiro e B é falso, então C é falso". Linha 3: "Se A é falso e B é verdadeiro, então C é falso". Linha 4: "Se A e B são ambos falsos, então C é falso".

A tabela-verdade para o **OR** é:

A	B	C	
-1	-1	-1	linha 1
-1	0	-1	linha 2
0	-1	-1	linha 3
0	0	0	linha 4

Na primeira linha, lê-se: "Se A é verdadeiro e B é verdadeiro, então C é verdadeiro". Nas linhas 2 e 3 lê-se: "Se A ou B for verdadeiro, então C é verdadeiro". A linha 4 significa: "Se A e B são ambos falsos, então C também é falso"

Apresentamos aqui um programa que usa **AND, OR** e **NOT**. Trata-se de uma versão simplificada de um programa que calcula a escala de salário correta para o candidato a um emprego.

```
10 INPUT"IDADE";IDADE
20 INPUT"NUMERO DE REFERENCIAS"
;REF
30 MAIOR18=(IDADE>=18)
40 QUAL=(REF>=5)
50 IF NOT MAIOR18 AND NOT QUAL
THEN PRINT "NAO QUALIFICADA"
60 IF(NOT MAIOR18 AND QUAL)OR(M
AIOR18  AND NOT QUAL) THEN PRIN
T "ESCALA 1 DE SALARIO"
70 IF MAIOR18  AND QUAL THEN PR
INT "ESCALA 2 DE SALARIO"
```

Digamos que o candidato tenha respondido **20** para idade e **4** para número de referências. Isso significa que **(IDADE>=18)** é verdadeiro, mas **(REF>=5)** é falso. A pessoa, portanto, é **MAIOR18** e **NÃO QUAL**ificada. Na linha 60, encontramos outro conjunto de condições que, caso satisfeitas, selecionam a primeira escala de salários. Poderíamos facilmente modificar o programa para testar mais condições, por exemplo: verificar se a pessoa tem mais de dois anos de experiência, etc.

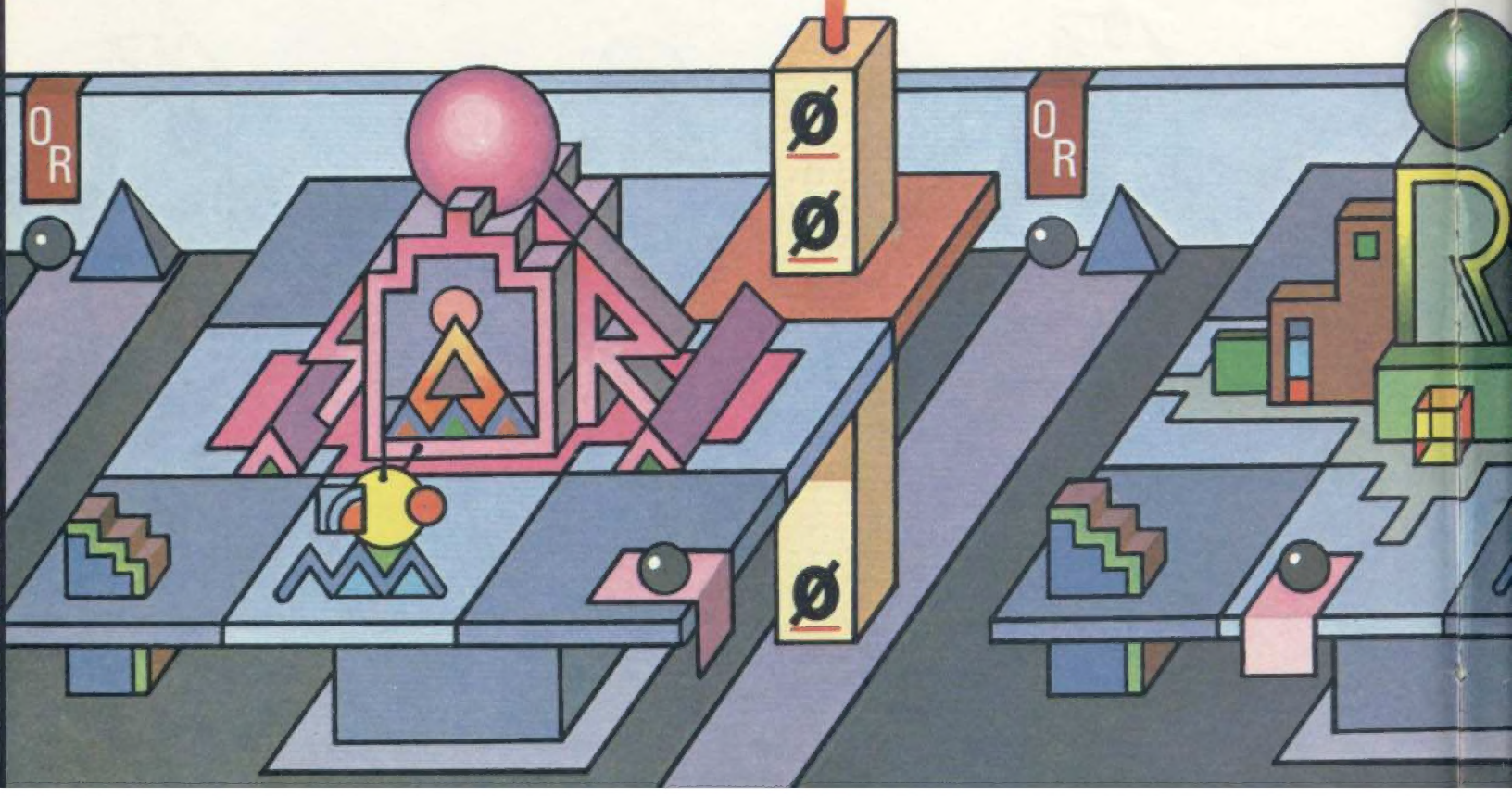
OPERAÇÕES NUMÉRICAS

O uso dos operadores lógicos não está limitado somente ao **IF...THEN**. Eles também podem ser usados com alguns tipos de operações numéricas. Talvez você tenha visto esse tipo de uso em programas anteriores, mas ficou sem saber o que estava acontecendo. Na verdade, eles nem sempre funcionam da maneira óbvia, como poderíamos esperar. Tente isto no modo direto:

```
PRINT 375 AND 47
```

Se você esperou que o resultado fosse 422 ou até mesmo 37547, errou. Na verdade, temos 39 como resultado. O que estaria acontecendo então? Somente se nos aprofundarmos um pouco no funcionamento dos computadores poderemos entender o que ocorre com o **AND** desse exemplo.

Em primeiro lugar, as expressões (375 e 47) são transformadas em números in-



teiros de dois bytes. Na forma binária, ficam assim:

375 em binário de dois bytes:
0000000101110111
47 em binários de dois bytes:
0000000000101111

Em seguida, o **AND** compara bit a bit todos os dezesseis bits, produzindo um "1" somente quando, no par de bits comparados, ambos forem "1". Da direita para a esquerda, os únicos pares de bits que satisfazem (produzem "1") são os de números 0,1,2 e 5. Isto resulta no número binário 0000000000100111 que, transformado para decimal, é 39. Portanto, 375 **AND** 47 é igual a 39.

Agora, tente em modo direto:

```
PRINT 375 OR 47
```

Como obtivemos 383? Lembre-se da propriedade do **OR**: produz-se "1" quando, no par de bits comparados, pelo menos um dos bits for "1". Observando novamente a forma binária de 375 e 47, vemos que os pares de bits de número 8, 6, 5, 4, 3, 2, 1 e 0 produzem "1" quando comparados pelo **OR**. Isto resulta no binário 0000000101111111, que equivale ao 383 decimal.

Com **OR** é possível obter o mesmo resultado, usando expressões diferentes. **PRINT 375 OR 75**, por exemplo, também resulta em 383. Caso queira verificar, faça o seguinte: converta para binário, compare pelo **OR** e converta de volta para decimal o resultado obtido.

O valor −1 | (armazenado como FFFFFFFF em hexadecimal — um arranjo de bits onde todos valem 1) não se altera quando comparado pelo **OR** com qualquer outro valor. Faça uma experiência, tentando no modo direto:

```
PRINT -1 OR 375
```

O resultado é −1.

Vejamos agora uma aplicação numérica para o **NOT**.

```
PRINT NOT 10.75 , NOT -11
```

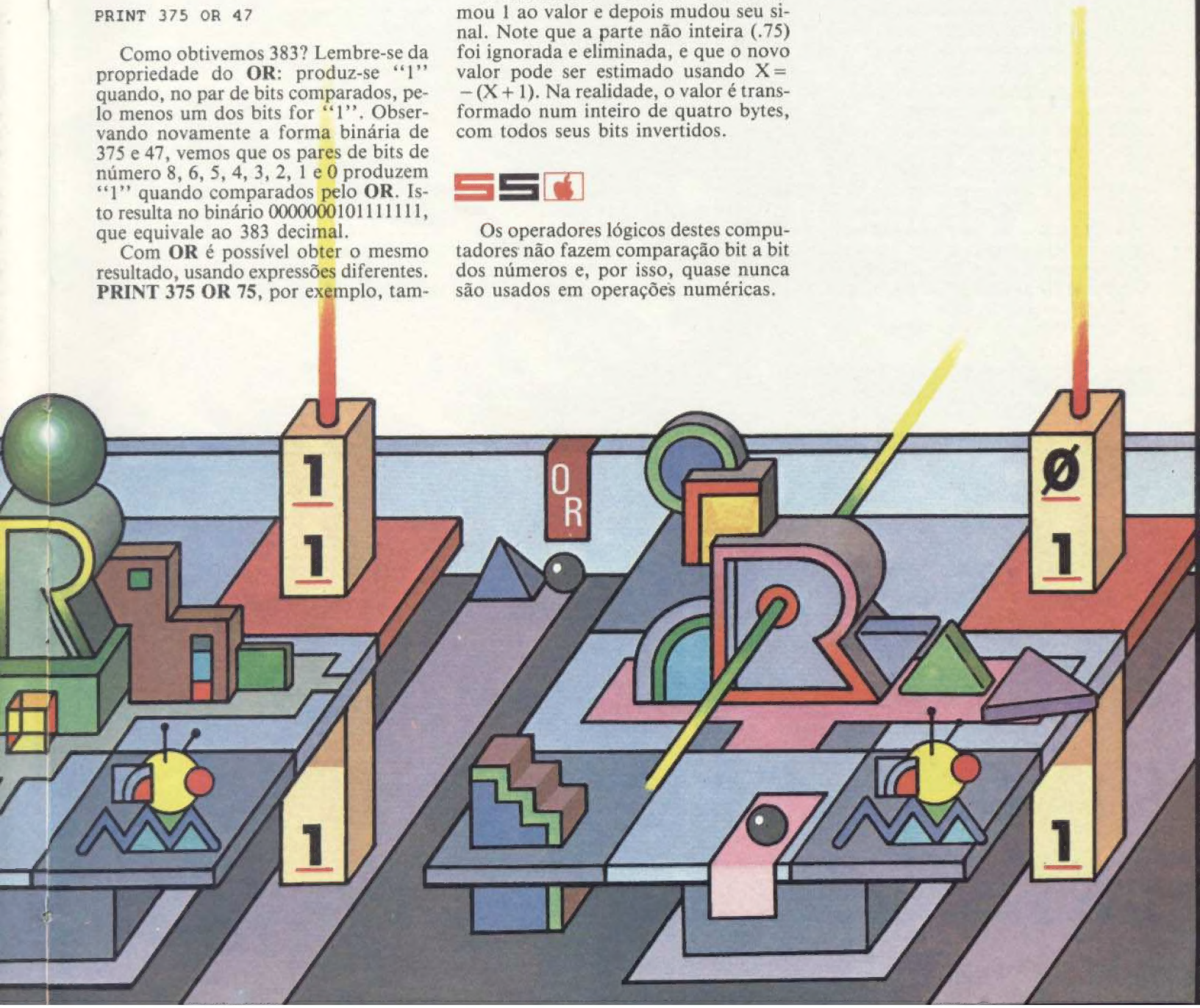
O resultado é −11 e 10. O **NOT** somou 1 ao valor e depois mudou seu sinal. Note que a parte não inteira (.75) foi ignorada e eliminada, e que o novo valor pode ser estimado usando X = −(X + 1). Na realidade, o valor é transformado num inteiro de quatro bytes, com todos seus bits invertidos.

Os operadores lógicos destes computadores não fazem comparação bit a bit dos números e, por isso, quase nunca são usados em operações numéricas.

EXISTE LIMITE PARA O TAMANHO DOS NÚMEROS USADOS EM *AND* E *OR*?

No TRS-Color e MSX, os números empregados em qualquer operação com **AND** ou **OR** têm que estar entre −32768 e +32767; caso contrário, teremos uma mensagem de erro.

No Spectrum, Apple e TK-2000 não é possível usar números nas operações **AND** ou **OR**.



OS OBJETOS DA AVENTURA

Chegou a hora de preencher o mundo ainda vazio de nossa aventura. Você verá aqui como acrescentar ao programa uma lista de objetos e, também, como manipulá-los.

Na última seção de Programação de Jogos, dispúnhamos de um conjunto completo de posições para nossa aventura e o jogador já estava capacitado a explorar todas elas. As movimentações do aventureiro, porém, até agora não têm propósito, uma vez que o mundo da aventura está vazio. Convém, assim, retomar o planejamento inicial e examinar o que se pretendia incluir em cada ponto.

Veremos, em seguida, como adicionar ao programa as rotinas que colocarão os objetos envolvidos na aventura em seus devidos lugares. Outras rotinas permitirão ao jogador carregar objetos consigo ou deixá-los de lado. Escreveremos, ainda, uma rotina para listar um inventário de todos os itens utilizados — ela será importante para o jogador, quando ele se deparar com um problema e precisar verificar exatamente a situação de cada objeto.

Use o **LOAD** e carregue o programa usado recentemente, deixando-o pronto para receber as novas rotinas.

OBJETOS

A máquina precisa saber três coisas sobre cada objeto da aventura: o número de posição onde foi inicialmente colocado, seu nome (ou descrição curta) e, finalmente, a descrição detalhada do local onde se encontra. As três informações são necessárias já que, primeiro, o computador deverá escolher um objeto adequado a cada posição. Depois, precisará contar ao aventureiro que objetos estão nesta posição — mesmo que use uma longa descrição. E, por último, terá que dispor de um nome para usar em instruções e na lista.

Os números de posição serão colocados em uma matriz, o nome do objeto em outra, e a descrição longa em uma terceira. As três matrizes serão manipuladas em paralelo pelo programa — cada elemento da matriz guardará um espaço equivalente de informação sobre o objeto: o primeiro conterá o número da posição, o segundo o nome do objeto e assim por diante. Adicione estas linhas ao seu programa:

- LINHAS **DATA** PARA A LISTA DE OBJETOS
- DESCRIÇÕES CURTAS E LONGAS
- COLOCAÇÃO DOS OBJETOS NAS POSIÇÕES CORRETAS
- MAIS VERBOS
- COMO PEGAR E LARGAR OBJETOS
- COMO MOSTRAR AO JOGADOR A LISTA DOS OBJETOS QUE CARREGA

S

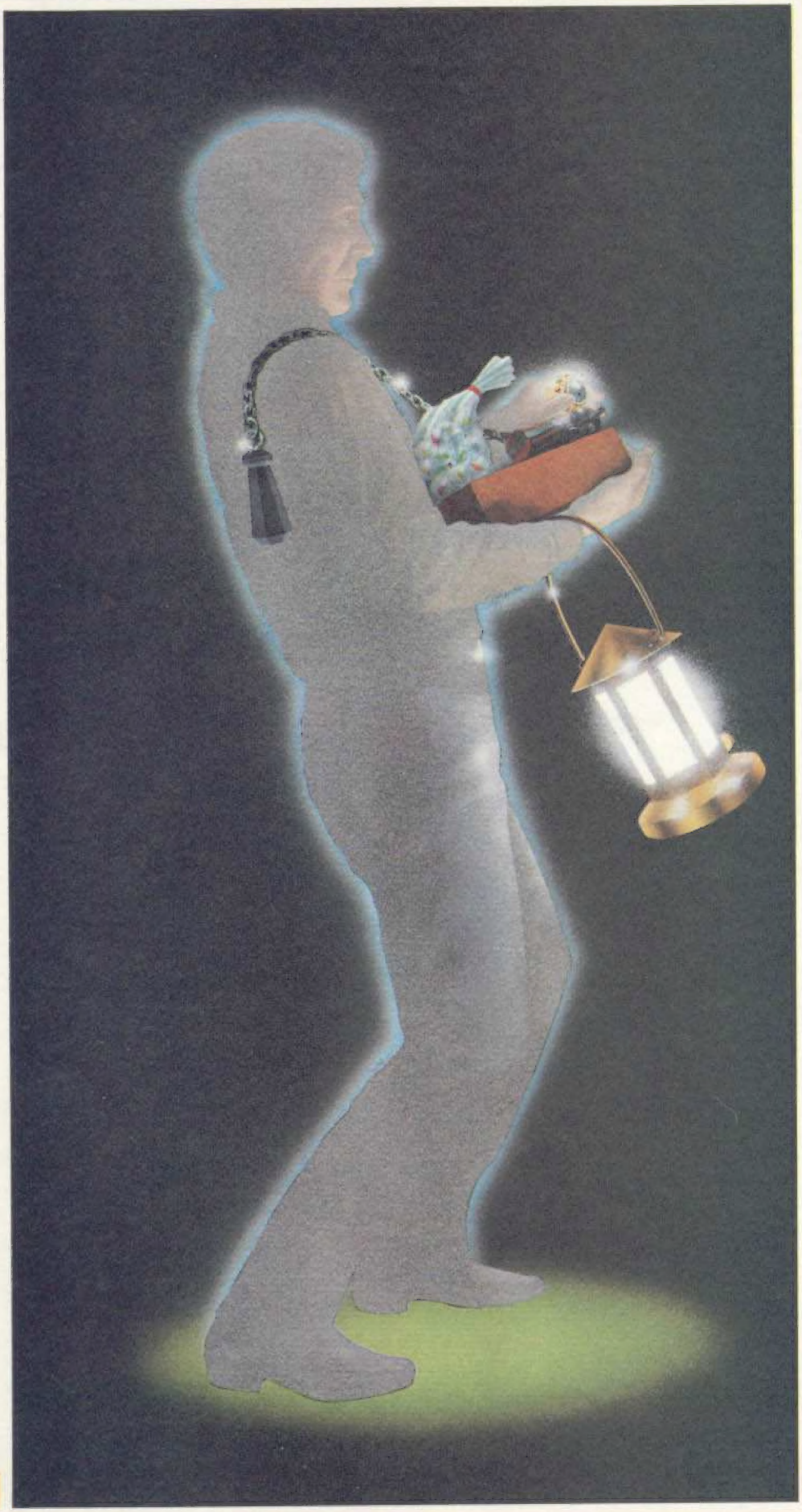
```
160 REM **PREPARA MATRIZES DE
OBJETOS**
170 READ NB
180 DIM B(NB): DIM B$(NB,14):
DIM S$(NB,40)
190 FOR I=1 TO NB: READ B(I),B
$(I),S$(I): NEXT I
200 DATA 7,4,"SACO","HA UM SAC
O DE BOLAS DE GUDE AQUI"
210 DATA 14,"TIJOLO","TEM UM T
IJOLO NO CHAO"
220 DATA 24,"CORRENTE","HA UMA
 CORRENTE PENDURADA SOBRE O TR
ONO"
230 DATA 0,"REVOLVER","TEM UM
REVOLVER NO CHAO"
240 DATA 0,"OLHO","UM OLHO CRA
VEJADO DE BRILHANTES ESTA NO C
HAO"
250 DATA 22,"LAMPADA","VOCE ES
TA DIANTE DE UMA LAMPADA"
260 DATA 0,"COLETOR","DE REPEN
TE SURGE UM COLETOR DE IMPOSTO
S"
```

T T

```
160 REM **PREPARA MATRIZES DE O
BJETOS**
170 READ NB
180 DIM OB(NB),OB$(NB),SI$(NB)
190 FOR I=1 TO NB:READ OB(I),OB
$(I),SI$(I):NEXT
200 DATA 7,4,SACO,HA UM SACO DE
 BOLAS DE GUDE AQUI
210 DATA 14,TIJOLO,TEM UM TIJOL
O NO CHAO
220 DATA 24,CORRENTE,HA UMA COR
RENTE PENDURADA SOBRE O TRONO
230 DATA 0,REVOLVER,TEM UM REVO
LVER NO CHAO
240 DATA 0,OLHO,UM OLHO CRAVEJA
DO DE BRILHANTES ESTA NO CHAO
250 DATA 22,LAMPADA,VOCE ESTA D
IANTE DE UMA LAMPADA
260 DATA 0,COLETOR,DE REPENTE S
URGE UM COLETOR DE IMPOSTOS
```

Cada linha entre 200 e 260 contém três partes dos dados referentes ao mesmo objeto. A linha 200 apresenta um dado a mais em sua lista. O número 7 — o primeiro do comando **DATA** — diz à máquina quantos conjuntos de dados existem.

Uma vez que o número 7 tenha sido



lido pela linha 170, três matrizes serão dimensionadas para este tamanho, pela linha 180. **OB** conterá a posição de cada objeto — um número da posição, ou 0, se o elemento ainda não existe (é como se fosse a tampa de um baú, que precisasse ser aberto a cada aventura), e −1, se está sendo levado pelo aventureiro. **OB$** conterá a descrição curta e **SI$** a descrição longa.

A linha 190 completará a matriz com os dados das linhas 200 a 260. Os comandos **DATA** estão arrumados em conjuntos de três elementos: um número de posição, a descrição curta do objeto e a descrição longa do objeto.

Para usar a mesma rotina em outras aventuras, não é necessário fazer muitas alterações nessa estrutura, pois, ajustando a primeira parte dos dados, automaticamente os comandos **FOR... NEXT** e as dimensões das matrizes estarão ajustados.

COMO POSICIONAR OS OBJETOS

O programa contém agora todas as informações sobre os objetos e as posições onde deverão ser colocados. A rotina seguinte exibe a descrição longa do objeto quando o jogador está no local adequado.

```
360 REM **COLOCA CADA OBJETO E
M SEU LUGAR**
370 FOR I=1 TO NB: IF B(I)=L
THEN  PRINT S$(I)
380 NEXT I
```

```
360 REM**COLOCA CADA OBJETO EM
SEU LUGAR**
370 FOR  I=1 TO NB:IF OB(I)=L T
HEN PRINT SI$(I)
380 NEXT
```

Neste estágio, faça uma pequena alteração nas linhas 330 e 340: troque **GOTO 400** para **GOTO 370**. As linhas 370 e 380 checam a matriz que guarda a posição do objeto. Se algum número de posição combinar com a posição corrente — **L** — uma descrição curta será exibida após a descrição da posição. Essa rotina pode ser usada em outras aventuras, sem alterações.

MAIS VERBOS

A aventura inclui objetos em diferentes posições mas, como a máquina até aqui não entende qualquer palavra ex-

ceto NORTE, SUL, LESTE e OESTE, o pobre aventureiro não pode fazer nada com eles. Imagine sua frustração ao se ver incapaz de pegar o tão desejado saquinho de bolas de gude, ou sem condições de se defender do fiscal da Receita! Precisamos fornecer ao computador um vocabulário de palavras que ele possa reconhecer, informando o que fazer com os objetos. Mais tarde, veremos como proceder se o jogador entrar uma palavra que não está no vocabulário programado.

Como o programa trata todas as palavras de direção como verbos, o melhor lugar para os verbos, que descrevem o que fazer com os objetos, é a matriz **R$** e, para os números correspondentes, **R**.

Precisaremos, no entanto, fazer algumas alterações na linha 130. Os limites do **FOR...NEXT** deverão ser mudados. Reescreva toda a linha ou use o editor da máquina para mudá-la. Seja qual for sua escolha, a linha 130 será agora:

```
130 FOR K=1 TO 19: READ R$(K),
R(K): NEXT K
```

```
130 FOR K=1 TO 19:READ R$(K),R(
K):NEXT
```

Agora, adicione as linhas 140 e 145:

```
140 DATA "NADAR","5","ESVAZIAR
","6","ACENDER","7","DESISTIR"
,"8","LISTAR","9","MATAR","10"
,"ATIRAR","10","AJUDAR","11"
145 DATA "PEGAR","2","APANHAR"
,"2","CARREGAR","2","COLOCAR",
"3","DEIXAR","3","LARGAR","3",
"PUXAR","4"
```

```
140 DATA NADAR,5,ESVAZIAR,6,ACE
NDER,7,DESISTIR,8,LISTAR,9,MATA
R,10,ATIRAR,10,AJUDAR,11
145 DATA PEGAR,2,APANHAR,2,CARR
EGAR,2,COLOCAR,3,DEIXAR,3,LARGA
R,3,PUXAR,4
```

A cada verbo corresponde um número. Verbos com o mesmo número possuem o mesmo significado e, portanto, o mesmo efeito. Planejamos o programa de modo que o computador reconheça, por exemplo, **PEGAR**, **APANHAR** e **CARREGAR**, evitando que o aventureiro gaste seu precioso tempo tentando descobrir qual desses verbos usar. Você pode facilmente adicionar suas próprias palavras às linhas de comando **DATA**: basta trocar o laço **FOR...NEXT** na linha 130 e colocar outro comando **DATA** após a linha 145. Você deverá fazer algumas alterações em outros locais do programa mas, nas próximas seções de Programação de Jogos, informaremos exatamente como proceder.

SELEÇÃO DAS ROTINAS ADEQUADAS

Depois de entrar todos os verbos na última rotina, o computador precisará de outras rotinas que o tornem capaz de agir conforme as instruções e de fazer com que o aventureiro carregue alguns objetos.

A sub-rotina que começa na linha 3010, por exemplo, define **V$**, **N$** e **I** (um número da matriz **R** que você já deve ter escrito).

Esta pequena rotina fará com que a máquina seja capaz de selecionar a rotina correta, de acordo com o valor de **I** — que é o significado da entrada do aventureiro.

O Spectrum não tem o comando **ON...GOTO**, usado em programas para outros computadores. As linhas do programa, portanto, têm que ser um pouco diferentes.

Já temos uma matriz, **G**, que contém números de linha para as descrições de posição. Os números de linha necessários às novas rotinas podem ser adicionados a esta matriz.

Por isso, a linha 30 ficou assim:

```
30 FOR N=1 TO 4: FOR M=1 TO
11: READ G(M,N): NEXT M: NEXT
 N
```

E é também pelo motivo acima que essa linha contém todos os números de linha que precisaremos (veja a explicação completa no artigo da página 270).

```
70 DATA 1010,1150,1240,1310,
1410,1460,1500,1360,1080,1550
,3110
```

Agora, adicione a rotina que solucionará a rotina correta, de acordo com o valor de **I**:

```
500 REM **SELECIONA OPCAO**
510 IF I=0 THEN  GOTO 520
520 PRINT '"EU NAO SEI COMO ";
V$: GOTO 370
```

Se a sub-rotina de verificação de instrução que começa na linha 3010 não encontrar uma combinação para **V$** em **R$**, **I** torna-se zero e a linha 510 faz com que a mensagem "EU NÃO SEI COMO" surja na tela. Se **I** tiver qualquer outro valor, a linha 515 encontra o número de linha correto na matriz **G** e executa um **GOTO**.

```
500 REM**SELECIONA OPCAO**
505 IF I=0 THEN GOTO 520
510 ON I GOTO 1010,1150,1240,13
10,1410,1460,1500,1360,1080,155
0,3110
520 PRINT:PRINT "EU NAO SEI COM
O ";V$:GOTO 370
```

Os números após o **ON...GOTO** na linha 510 iniciam as diversas rotinas. Cada valor de **I** é um verbo diferente ou um grupo de verbos. Se **I = 10**, por exemplo, a rotina "MATAR" será selecionada — e, sendo o décimo número na linha, ela começará na linha 1550.

Se a sub-rotina de verificação de instrução que começa na linha 3010 não encontrar uma combinação de **V$** em **R$**, **I** torna-se 0. Nesse caso, o **ON...GOTO** na linha 510 não terá nenhum efeito. A mensagem na linha 520 será exibida na tela.

COMO PEGAR OS OBJETOS

Já temos uma rotina para quando **I** for igual a 1, o que ocorre quando o aventureiro dá uma ordem. Ela está entre as linhas 1010 e 1060.

Quando **I = 2**, o aventureiro digitou uma rotina "PEGAR" — ou seja, as palavras **PEGAR**, **APANHAR** ou **CARREGAR**. Esta rotina, apresentada a seguir, permitirá ao aventureiro pegar e conservar consigo qualquer objeto que esteja na posição.

```
1140 REM **PEGAR**
1150 FOR G=1 TO NB
1160 IF N$=B$(G, TO LEN N$) THE
N  GOTO 1190
1170 NEXT G
1180 PRINT N$;"???": GOTO 330
1190 IF B(G)=-1 THEN  PRINT "VO
CE PEGOU": GOTO 330
1200 IF B(G)<>L THEN  PRINT "NA
O ESTA AQUI": GOTO 330
1210 PRINT "OK": LET B(G)=-1
1220 GOTO 330
```

```
1140 REM**PEGAR**
1150 FOR G=1 TO NB
1160 IF INSTR(OBS$(G),NS)=1 THEN
 GOTO 1190
```

```
1170 NEXT
1180 PRINT N$;"???":GOTO 330
1190 IF OB(G)=-1 THEN PRINT "VO
CE PEGOU":GOTO 330
1200 IF OB(G)<>L THEN PRINT "NA
O ESTA AQUI":GOTO 330
1210 PRINT "OK":OB(G)=-1
1220 GOTO 330
```

```
1140  REM ** PEGAR **
1150  FOR G = 1 TO NB
1160  IF N$ =  LEFT$ (OB$(G), L
EN (N$)) THEN 1190
1170  NEXT
1180  PRINT N$;" ??"; GOTO 330
1190  IF OB(G) =  - 1 THEN  PRI
NT "VOCE PEGOU": GOTO 330
1200  IF OB(G) <  > L THEN  PRI
NT "NAO ESTA AQUI": GOTO 330
1210  PRINT "OK":OB(G) =  - 1
1220  GOTO 330
```

As linhas 1150 a 1170 procuram a matriz contendo a descrição curta — **B$**, no caso do Spectrum, e **OB$** nos demais computadores — do objeto que o aventureiro chamou. Se este é encontrado, o programa pula para a linha 1190. Caso contrário, a linha 1180 exibe o nome do objeto que o aventureiro digitou, seguido de pontos de interrogação.

Depois de encontrar o nome do objeto, duas verificações são feitas. A linha 1190 checa o elemento da matriz de posição do objeto — **B** ou **OB** —, para saber se ele já foi levado. Em caso afirmativo (o valor do elemento da matriz é −1), a mensagem "VOCÊ PEGOU" será exibida.

A linha 1200 verifica se o objeto está presente, examinando novamente a posição da matriz. Se ele não estiver presente, o programa exibirá a mensagem "NÃO ESTÁ AQUI". Você poderá trocar essa mensagem, se ela não servir para a sua aventura.

Caso o objeto não tenha sido levado e se encontre na mesma posição que o aventureiro, a linha 1210 exibirá "OK" e o elemento na matriz de posição de objetos será mudado para −1.

COMO DEIXAR OBJETOS DE LADO

A rotina "DEIXAR" faz o contrário. Ela permite que o jogador abandone qualquer dos objetos que pegou.

```
1230 REM **DEIXAR**
1240 FOR G=1 TO NB
1250 IF N$=B$(G, TO LEN N$) THE
N  GOTO 1270
1260 NEXT G: PRINT N$;"???": GO
TO 330
1270 IF B(G)<>-1 THEN  PRINT "V
OCE NAO PODE LARGAR O QUE NAO T
EM": GOTO 330
1280 PRINT "OK": LET B(G)=L
1290 GOTO 330
```

```
1230 REM**DEIXAR**
1240 FOR G=1 TO NB
1250 IF INSTR(OB$(G),N$)=1 THEN
 1270
1260 NEXT:PRINT N$;"???":GOTO 3
30
1270 IF OB(G)<>-1 THEN PRINT "V
OCE NAO PODE LARGAR O QUE NAO T
EM":GOTO 330
1280 PRINT "OK":OB(G)=L
1290 GOTO 330
```

```
1230  REM ** DEIXAR **
1240  FOR G = 1 TO NB
1250  IF N$ =  LEFT$ (OB$(G), L
EN (N$)) THEN 1270
1260  NEXT : PRINT N$;" ??": GO
TO 330
1270  IF OB(G) <  >  - 1 THEN
PRINT "VOCE NAO PODE LARGAR O Q
UE NAO TEM": GOTO 330
1280  PRINT "OK":OB(G) = L
1290  GOTO 330
```

Seu funcionamento se assemelha muito ao da rotina "PEGAR", vista anteriormente. A matriz de descrição curta é mais uma vez procurada — agora nas linhas 1240 a 1260. Se o objeto pedido estiver na matriz, a linha 1270 verifica se está sendo levado pelo aventureiro. Se não estiver, o computador exibirá a mensagem "VOCÊ NÃO PODE LARGAR O QUE NÃO TEM".

Se o aventureiro estiver carregando o objeto, a linha 1280 exibe "OK" e o elemento apropriado na matriz de posição de objetos — **OB** ou **B** — é ajustado. Ele toma, então, o número da posição corrente — L —, já que −1 significa que o objeto estava sendo carregado.

A LISTAGEM DO SAQUE

O aventureiro distraído ficará muito grato ao obter uma lista de todos os objetos que carrega.

Aqui está uma rotina que faz exatamente isso:

```
1070 REM **LISTAR**
1080 PRINT "VOCE TEM ";: LET IN
=0
1090 FOR G=1 TO NB
1100 IF B(G)=-1 THEN  PRINT TAB
 10;B$(G): LET IN=IN+1
1110 NEXT G
1120 IF IN=0 THEN  PRINT "NADA"
1130 GOTO 330
```

```
1070 REM**LISTAR**
1080 PRINT "VOCE TEM ";:IN=0
1090 FOR G=1 TO NB
1100 IF OB(G)=-1 THEN PRINT TAB
(10)OB$(G):IN=IN+1
1110 NEXT
1120 IF IN=0 THEN PRINT "NADA"
1130 GOTO 330
```

```
1070  REM ** LISTAR **
1080  PRINT "VOCE TEM ";:IN = 0
1090  FOR G = 1 TO NB
1100  IF OB(G) =  - 1 THEN  PRI
NT  TAB( 10);OB$(G):IN = IN + 1
1110  NEXT
1120  IF IN = 0 THEN  PRINT "NA
DA"
1130  GOTO 330
```

"VOCÊ TEM" será exibido pela linha 1080, antes que se inicie a listagem dos objetos. O laço **FOR...NEXT** checa, um a um, os elementos da matriz de posições de objetos. Dessa vez, os elementos importantes são os que contêm −1, indicando que o objeto está sendo carregado. Se o valor de qualquer um dos elementos for −1, a descrição curta do objeto será, então, exibida. O contador do inventário I será incrementado de 1.

Se nenhum objeto estiver sendo carregado, o contador **IN** continua em zero e a linha 1120 exibe "NADA", em vez da lista de objetos.

As rotinas "PEGAR", "DEIXAR" e "LISTAR" podem ser usadas como estão, se **NB** for definido em uma rotina anterior.

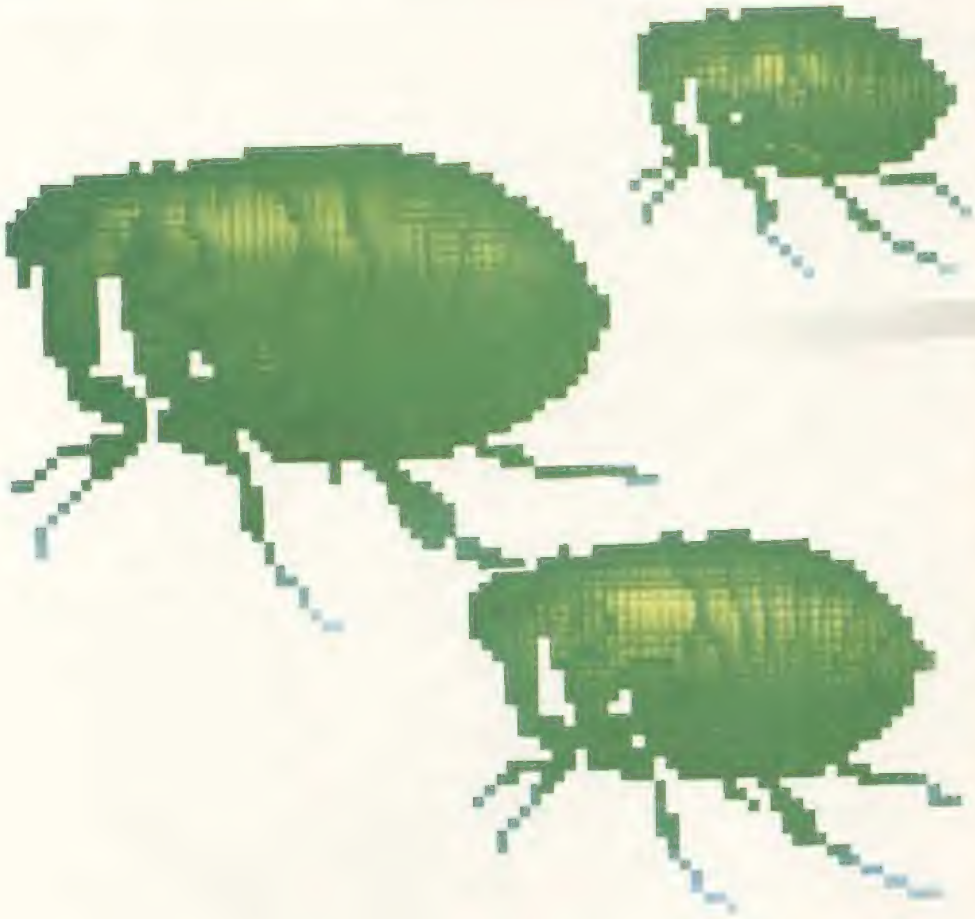
Agora, o programa está pronto para receber as rotinas finais, que serão apresentadas num próximo artigo. Elas se referem ao fiscal da Receita, ao tijolo, à lâmpada, à procura do globo ocular, ao término da aventura e, finalmente, à instrução descrevendo o objeto da busca.

Se você fizer uma experiência, executando o programa neste estágio, verá que, enquanto parte dele funciona, alguma coisa estranha acontece. A razão para isso é que o programa requer um número de rotinas que ainda não existe. Se você entrar certas palavras, o programa tentará desviá-las para linhas inexistentes.

COMO EVITAR E DETECTAR ERROS

- MENSAGENS DE ERROS
- ORIENTAÇÕES OBSCURAS
- OS ERROS MAIS COMUNS
- TESTE SUA HABILIDADE EM DETECTAR ERROS

Nenhum programa está livre de erros. Aprenda a localizá-los e habitue-se a corrigi-los na medida em que aparecem. Esta é, sem dúvida, a melhor maneira de evitar problemas mais sérios.



Nenhum programa, de qualquer tamanho, que esteja sendo desenvolvido ou simplesmente copiado de uma listagem, está completamente livre de erros — os "grilos". Estes aparecem pelas mais diversas razões e comprometem o programa em graus diferentes.

Os erros mais sérios podem impedir que um programa seja rodado. Outros, simplesmente, permanecem ocultos, até que certa rotina ou seqüência de entradas os revele. Por exemplo, em um jogo de aventuras, tudo pode funcionar perfeitamente até que o jogador tome um determinado caminho, carregando uma faca — e cai num buraco que não deveria estar ali. Ou, em um programa de contabilidade, pode-se de repente encontrar um erro enorme, que afeta apenas as entradas feitas na segunda semana de dezembro.

O primeiro tipo de erro deve ser completamente eliminado antes da utilização do programa.

E o segundo? Bem, o ideal seria procurá-los sistematicamente, para evitar surpresas. A melhor maneira de fazê-lo é testar cuidadosamente o programa — o que inclui tentar o inesperado, como entrar uma seqüência "impossível" de entradas. Para auxiliar nesse tipo de tarefa, existem rotinas muito úteis, especiais para detecção de erros. Logo falaremos sobre elas.

MENSAGENS DE ERRO

Todos os computadores domésticos produzem mensagens de erro, de um tipo ou de outro. Você já deve ter encontrado várias delas, desde que começou a utilizar o computador.

Tais mensagens variam de códigos simples de números e letras a descrições completas que deixam poucas dúvidas sobre a natureza do erro — ainda que, às vezes, não apontem diretamente o próprio erro.

Maiores detalhes sobre as mensagens de erro podem ser encontrados no manual do seu computador. Consulte-o sempre que não compreender o significado exato de alguma delas. Só depois comece a trabalhar na correção do erro.

"DEPURE" O PROGRAMA

Para evitar problemas maiores, é muito importante localizar e corrigir cada erro na medida em que aparece. Não deixe um erro permanecer em um programa, mesmo que este pareça rodar satisfatoriamente.

Caso contrário, sempre haverá uma chance de que o erro apareça mais tarde, numa hora crítica. A conseqüência poderá ser um desastre para o programa, o que resultará em muita digitação extra, ou — pior ainda — na perda dos dados disponíveis, se o programa for colocado em uso efetivo. No final, você não terá mais a oportunidade de resolver o problema.

Corrigir os erros, ou seja, "depurar" o programa, pode se tornar uma tarefa terrivelmente complicada se você não se empenhar em isolar cada problema de modo sistemático. Ao digitar um programa pronto ou ao desenvolver o seu próprio, é muito provável que cometa mais de um erro. Trate cada um individualmente — ou seja, localize e corrija o primeiro deles e só depois se dirija ao seguinte.

TABELA DE LOCALIZAÇÃO

Esta é uma linha de mensagens de erros e comunicados para cada computador. Quando uma entrada for precedida por um asterisco (*), o erro provavelmente se encontra na linha cujo número está indicado na mensagem ou, então, é o resultado imediato de uma entrada direta ou de uma atividade (tal com SAVE). Sempre existem exceções, especialmente quando uma variável, que é a causa de um erro em determinada linha, tem seu valor designado em uma linha previamente executada.

Quando digitar os programas, seja cuidadoso ao incluir espaços.

*NEXT sem FOR, variável inexistente, Erro de índice, *Memória lotada, *Excede área de vídeo, Número excede limite, RETURN sem GOSUB, *Fim de arquivo, *Stop executado, Argumento inválido, Inteiro excede limite, *Erro de sintaxe, *BREAK-CONT repete, Fim de dados, *Nome inválido, *Sem memória disponível, STOP em INPUT, FOR sem NEXT, Periférico inválido, *Cor inválida, BREAK-CONT prossegue, RAMTOP válido, *Comando perdido, *Canal inválido, FN sem DEF, Erro de parâmetro, *Erro de leitura.*

*/O, AO, BS, *CN, *DD, *DN, DS, FC, FD, FM, *ID, IE, *IO, LS, NF, NO, OD, OM, OS, OV, RG, *SN, *ST, *TM, *UL.*

*"CONT" ILEGAL, *DIVISÃO POR ZERO, *DIRETO ILEGAL, *VALOR ILEGAL, "NEXT" SEM "FOR", FIM DE DATA, FALTA MEMÓRIA, *FÓRMULA COMPLEXA, *S/ESPAÇO, *REDIMENSIONAR, "RETURN" SEM "GOSUB", "STRING" LONGA, ÍNDICE ILEGAL, *SINTAX ERRO, *TIPO INCOMPAT, *LINHA INDEFINIDA, FUNÇÃO INDEF.*

*"NEXT" SEM "FOR", *ERRO SINTAXE, "RETURN" SEM "GOSUB", SEM "DATA", FUNÇÃO ILEGAL, *OVERFLOW, FALTA MEMÓRIA, *Nº LINHA INEXISTENTE, ÍNDICE FORA DO LIMITE, "DIM" REDEFINIDO, *DIVISÃO POR ZERO, DIRETO ILEGAL, TIPO DESIGUAL, *FALTA ÁREA *"STRING", *"STRING" LONGA, *"STRING" COMPLEXA, NÃO CONTÍNUO! FUNÇÃO NÃO DEFINIDA, ERRO/PERIFÉRICO, ERRO/VERIF., "RESUME", "RESUME" SEM "ERROR", *FALTA OPERANDO, *LINHA MUITO LONGA.*

LOCALIZAÇÃO DOS ERROS

Não se esqueça de que o erro mais simples é, em geral, o mais difícil de ser isolado, e — ao contrário do que poderá pensar — quase sempre é a única causa da dificuldade em rodar um programa adequadamente.

Muitos problemas podem ser evitados com a utilização de um conjunto de rotinas para depurar os programas. Existem algumas técnicas bem simples e caminhos muito eficazes.

Para começar, muitas mensagens de erro apontam diretamente para a linha na qual ocorre o erro. Isso inclui o mais comum deles — **ERRO DE SINTAXE** —, além de vários outros (veja a tabela). Alguns, porém, são menos evidentes, e indicam erros em linhas que estão perfeitamente corretas. Você pode obter uma mensagem como **E Fim de dados 10:2** (o exemplo é do Spectrum, mas outros computadores apresentam mensagens semelhantes). Aparentemente, trata-se de uma indicação de que o erro se encontra na segunda instrução da linha 10. Na verdade, a segunda instrução da linha 10 diz ao computador para ler alguns dados, o que poderia aparecer também na linha 200 ou até mesmo na linha 2000. Por outro lado, talvez o erro estivesse nas linhas de dados, e não na linha 10.

Esse tipo de erro é, evidentemente, muito mais difícil de se localizar, já que não existe nenhuma indicação precisa de onde se encontra o verdadeiro problema.

As mensagens de erro, que aponta a linha, revelam apenas que o problema deve estar relacionado à maneira como se executou uma entrada em determinada linha do programa.

Inicie efetuando uma listagem do programa a partir de um ponto antes do número da linha sugerida na mensagem de erro. Algumas vezes, é útil listar a própria linha, separadamente e um pouco além das outras, se isso for possível em seu computador.

Antes de mais nada, verifique se você não cometeu nenhum tipo de *erro literal* — ou seja, se escreveu algo de modo errado ou se utilizou uma letra no lugar de um número (ou vice-versa). Seja especialmente cuidadoso em relação aos espaços e à pontuação. E procure ver se falta algum caractere — é muito fácil, por exemplo, esquecer um parêntese em uma seqüência que utilize vários deles. Examine com atenção as palavras-chave, letra por letra, pois também é bastante comum a inversão da ordem dos caracteres.

Os erros literais costumam ser uma causa freqüente de problemas, perturbando o desenvolvimento do trabalho tanto de iniciantes quanto de especialistas.

LINHA POR LINHA

Quando você sabe que o erro está em determinada linha, e esta contém duas instruções, precisará descobrir qual delas apresenta o problema. O melhor método para isso é colocar uma instrução **STOP** em uma nova linha, imediatamente após a que foi indicada pela mensagem de erro. Em seguida, entre uma instrução **REM** bem no início da última declaração e rode novamente o programa — se nenhum desastre tiver acontecido até este ponto, é claro. Caso o erro de sintaxe não apareça, você saberá que ele se encontra na última declaração.

Pode-se utilizar um método parecido para descobrir, entre algumas linhas, a que contém o erro. Basta inserir sucessivamente, entre cada linha, uma linha extra com a instrução **STOP**, como mais tarde explicaremos em detalhe.

O método, porém, não pode ser ampliado com segurança para as linhas com instruções múltiplas, pois qualquer coisa antes do **REM** e depois da próxi-

ma linha do programa será ignorada. Nestes casos, deve-se dividir a linha em seus componentes e verificar isoladamente um por um. Cada instrução pode ser colocada em uma linha adicional. O procedimento é simples e bastante útil, sobretudo quando a situação parece muito confusa e obscura.

Rode o programa mais uma vez para ver qual das novas linhas do grupo causa a mensagem de erro e retire-a do programa.

PARTE POR PARTE

Declarações individuais muito longas apresentam um tipo diferente de problema e, talvez, a melhor maneira de unilas seja determinar o valor real de cada parte delas.

Mais uma vez, considere a hipótese de ter cometido um erro banal, antes de buscar interpretações complicadas. Se possível, reescreva a linha defeituosa do programa, de modo que obtenha um número de linhas separadas que possam ser tratadas individualmente. Esta é uma boa razão para se deixar espaços entre os números de linhas.

Onde você não puder fazer isto, "exploda" mentalmente a instrução e pergunte-se o que cada entrada está fazendo naquele determinado ponto do programa. Tente calcular quais os valores que foram obtidos para todas as variáveis em uso neste exato ponto.

Com o erro de sintaxe, os valores de qualquer variável em ação são irrelevantes — eles não poderão ser a causa do problema. Assim, mude a linha à vontade, mesmo que isso limpe as variáveis.

Se estiver lidando com um problema muito obscuro, o valor (ou valores) real da variável poderá fornecer os indícios. Utilize o computador para imprimir esses indícios no modo direto (imediato), antes de efetuar qualquer modificação no programa.

Simultaneamente, verifique o nome da variável, para se certificar de que está correto. Suponhamos que você habitualmente utilize a variável **D** para um loop de atraso de tempo. É bem provável que a introduza no programa sem querer, quando estiver copiando uma listagem e se deparar com um loop parecido, mas que utiliza uma variável diferente — um **T**, por exemplo.

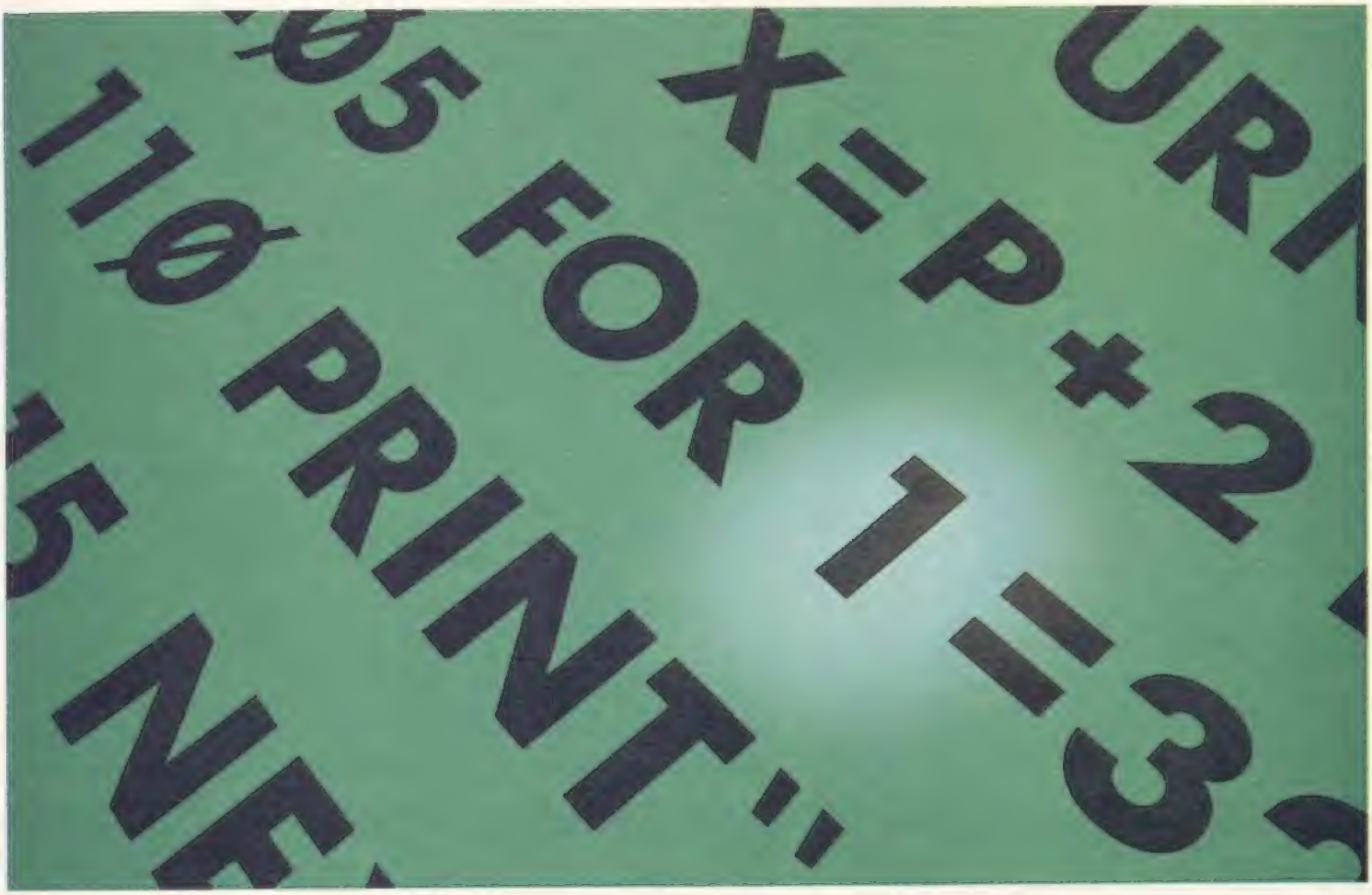
ORIENTAÇÕES OBSCURAS

Muitas mensagens de erro não fornecem indicações claras do número da linha errada. Em vez disso, simplesmente indicam onde um erro teve algum efeito. Quando isso ocorre, pode ser difícil apontar a causa com precisão.

Mas existem exceções. Como no exemplo acima, as mensagens **DATA** de erro são exibidas em uma linha que contém uma instrução incluindo **READ** quando, na verdade, o próprio erro se encontra na linha de instrução **DATA**.

Outras mensagens são menos evidentes (veja a tabela). No entanto, a maioria delas refere-se a erros que ocorrem antes do número das linhas apostrofadas na execução do programa. Essa é uma distinção importante, pois o erro pode estar, de fato, em uma sub-rotina situada antes ou depois da linha indicada na declaração de erro. Assim, uma variável poderá apanhar um valor incorreto bem antes de identificá-lo.

A melhor maneira de lidar com erros menos óbvios é examinar a ação do programa. Inicie imprimindo o valor de cada variável. Se você possuir uma impressora conectada ao computador, poderá



efetuar uma cópia deles. Mas é fácil utilizar o comando direto **PRINT** ou qualquer abreviação adequada, seguida pelo nome da variável — tal como **PRINT V** — e anotar os valores.

Examine, depois, como as variáveis funcionam em relação uma à outra. Novamente, poderá ser bastante útil dividir uma linha muito longa do programa em linhas mais curtas, cada qual contendo uma única instrução.

Pesquise cada variável a partir desse ponto do programa, para comparar seu valor real (o número que você anotou) com o que poderia ter se o programa rodasse corretamente.

Em um programa muito extenso, o trabalho será difícil e demorado. Parte dele poderá ser evitada se você rodar o programa em vários estágios distintos — imediatamente antes e depois de um determinado conjunto de entradas, por exemplo. Observe, então, as modificações dos valores das variáveis.

A não ser que você esteja familiarizado com o uso das teclas <**RUN/STOP**> ou <**BREAK**> do seu computador, é aconselhável introduzir no programa linhas provisórias contendo a declaração **STOP**. Mas note que, com esse procedimento, você limpará a memória e, assim, deverá rodar o programa mais uma vez — o que nem sempre será possível.

Quando o programa parar, imprima os valores das variáveis e, em seguida, tecle a instrução para continuar até o próximo **STOP**.

ERROS COMUNS

Entre as causas mais comuns de erros de programa estão as falhas cometidas no código de entrada, a partir de listagens impressas.

O problema específico é gerado sobretudo pela confusão entre caracteres parecidos: casos típicos são as letras l em tipo minúsculo, I em tipo maiúsculo e o número 1, assim como a letra O e o número 0.

É fácil também confundir, sobretudo em listagens de pouca qualidade, dois pontos com ponto e vírgula, e ponto final com vírgula. O ponto e vírgula é utilizado para a formatação da exibição, e o uso acidental de dois pontos (sinal que indica o fim de uma declaração), como no exemplo abaixo, poderá resultar simplesmente em uma mensagem de "ERRO DE SINTAXE":

```
95 INPUT "ENTRE SEU NOME":N$
```

A confusão entre o ponto final e a vírgula, porém, é ainda mais difícil de se identificar. Veja estas duas linhas:

```
1000 DATA 12.4,6,7,8,13,1.7
1000 DATA 12.4,6,7,8.13,1.7
```

Tanto uma quanto outra pareceriam corretas, até mesmo depois de um minucioso exame. Note que existem seis itens de dados na primeira linha, mas apenas cinco na segunda.

Este poderia ser um indício de erro, caso as outras declarações **DATA** contivessem um número idêntico de itens. Incidentalmente, portanto, teríamos um método simples de realizar pelo menos uma verificação.

Se uma mensagem de erro exibiu "FIM DE DADOS", não existem itens de dados suficientes; portanto, a segunda linha poderia estar incorreta.

A presença de muitos itens, por sua vez, não provoca uma mensagem de erro; simplesmente assinala os valores incorretos para a variável **READ** ou, ainda, o valor incorreto para algumas das variáveis, se uma outra linha de dados for curta.

Uma mensagem de erro pode também resultar em uma linha **READ** ou uma linha de cálculo, se o tipo incorreto do valor for assinalado para uma variável **READ**.

O uso da letra O em vez do número 0 — ou vice-versa — pode acarretar um efeito idêntico.

ESPAÇOS

Deve-se deixar espaços em determinados lugares do programa, mas eliminá-los totalmente em outros — o problema, quase sempre, é reconhecer que algo tão "transparente" seja a causa de um problema.

Os espaços utilizados por razões estéticas — para separar, por exemplo, uma declaração impressa de outra ou tornar as listagens um pouco mais claras — não ocasionam erros se forem omitidos. Mas suspeite deles se a exibição na tela parecer achatada ou tiver partes superpostas.

Uma mensagem de erro ocorre se, acidentalmente, você incluir um espaço entre certas palavras-chave e o argumento que as segue colocado entre parênteses. Por exemplo: **TAB(n)** está correto; **TAB (n)**, não. As condições que determinam erros relacionados a espaço variam de um computador para outro, mas vale a pena verificar esse aspecto se não detectar falhas de outro tipo.

ERROS ESPECÍFICOS

Cada computador apresenta um tipo peculiar de erro, quase sempre relacionado a imperfeições — e não erros, propriamente — no sistema de operações ou no próprio hardware. Com freqüência, a falha se encontra no próprio BASIC e certas palavras-chave não se comportam adequadamente sob determinadas circunstâncias. Algumas das mais comuns estão detalhadas aqui, e serão apontadas em INPUT sempre que possam causar problemas inesperados, caso não se tome o devido cuidado.

FAÇA UM TESTE

Eis aqui um teste para a sua habilidade em detectar erros e uma oportunidade para colocar em prática tudo o que aprendeu. Os programas que se seguem estão cheios de erros — do tipo que poderia ser introduzido durante a cópia de uma listagem ou na adaptação de um programa sobre cuja ação não se tem um conhecimento completo. Eles oscilam de erros simples de digitação e sintaxe a erros na estrutura do próprio programa.

A versão correta do programa foi publicada na página 195 de INPUT. Adicionamos aqui uma linha extra para fazer com que o programa seja rodado novamente. Mas não olhe o programa original até que tenha elaborado o seu próprio caminho para localizar os erros.

O programa é um exemplo bem curto de como se deve distribuir os objetos entre os compartimentos de um jogo de aventuras. A lista dos objetos é lida a partir de uma lista de dados organizados em um conjunto; os números dos compartimentos estão armazenados em outro conjunto. A parte principal do programa — as linhas 70 a 100 — designa, aleatoriamente, um objeto para cada compartimento, verifica se todos foram utilizados e se há apenas um em cada compartimento. A linha 110 simplesmente imprime o resultado. Pelo menos é isto o que o programa deveria fazer se estivesse correto.

É provável que você identifique vários erros imediatamente, por meio da simples leitura do programa. Tente consertar todos os que puder e, então, quando estiver com o programa limpo, digite-o em seu computador e experimente rodá-lo. Certamente restarão alguns erros escondidos que você não identificou na primeira vez.

Se não conseguir resolver tudo, volte à página 195; de qualquer maneira, a versão correta da última linha deverá ser elaborada por você. E tenha o cuidado de não introduzir novos erros quando estiver corrigindo os outros!

```
10 LET g=14
20 DIM a$(g,7): DIM a(g)
30 FOR z=1 TO g
40 READ a$(g)
50 LET a(z)=z
70 FOR x=g TO 1 STEP -I
80 LET q=INT (RND*x)+1
90 LET t=a(x): LET a(x)=a(q):
 LET a(q)=T
100 NEXT x
110 FOR t=1 TO g: PRINT "A sal
a;t;tem";a$(a(t)): NEXT t
120 DATA corda,"espada","espan
ador","faca","revolver","chave
""tocha","carro","lanterna","m
achado","bomba","livro","aerom
odelo","robo"
130 GOTO 10
```

```
10  LET G = 14
20  DIM A$(G),A(G)
30  FOR Z = 1TOG
40  READ A$(G)
50  LET A(Z) = Z
70  FOR X = G TO 1 STEP  - I
80  LET Q =  INT ( RND (1) * X)
 + 1
90  LET T = A(X): LET A(X) = A(
Q); LET A(Q) = T
100  NEXT X
110  FOR T = 1 TO G: PRINT "A S
ALA";T"TEM";A$(A)(T)): NEXT T
120  DATA  CORDA,ESPADA,ESPANA
DOR,FACA,ARMA,CHAVE,TOCHA,CARRO
,LIQUIDIFICADOR,SOFA,BOMBA,LIVR
O,AEROMODELO,ROBO
130  GOTO 10
```

Para o MSX, adicione a linha abaixo:

```
5 R =  RND ( - TIME)
```

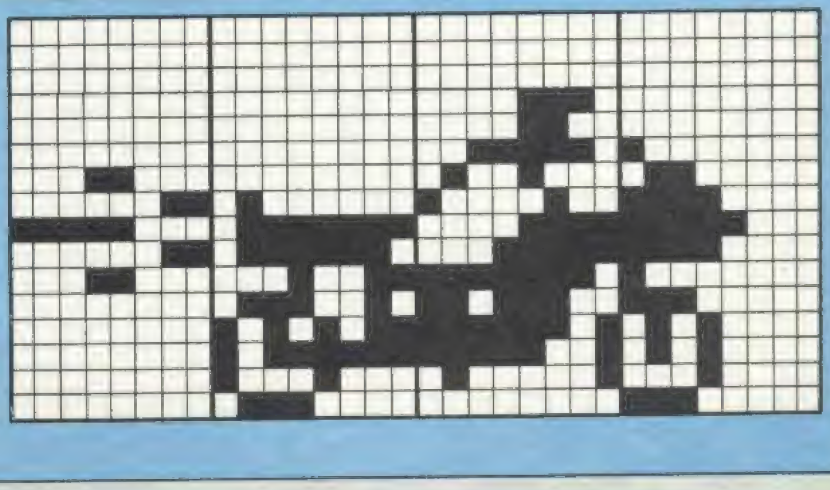
```
10 LET G=14
20 DIM A$(G),A(G)
30 FOR Z=1 TO G
40 READ A$(G)
50 A(Z)=Z
70 FOR X=G TO 1 STEP-I
80 Q=RND(X)
90 T=A(X):A(X)=A(Q);A(Q)=t
100 NEXT X
110 FOR T=1 TO G:PRINT"NA SALA"
;T"HA"A$(A)(T)):NEXT T
120 DATE CORDA,ESPADA,ESPANADOR
,FACA,ARMA,CHAVE,TOCHA,CARRO,LA
NTERNA,MACHADO,BOMBA,LIVRO,AERO
MODELO,ROBO
```

FIGURAS MÓVEIS

O ZX-81 não tem muitos recursos gráficos. O Apple, embora os tenha com boa qualidade e resolução, define as figuras móveis por meio de um processo bastante complicado.

Com o código de máquina podemos obter alguns efeitos visuais no ZX-81. Um programa editor facilitará a criação de figuras móveis no Apple.

Enquanto outros micros usam uma codificação bem simples para criar figuras em alta resolução — números binários, onde "1" e "0" correspondem a pontos "acesos" e "apagados" —, o Apple tem um código complexo, que não explicaremos agora.

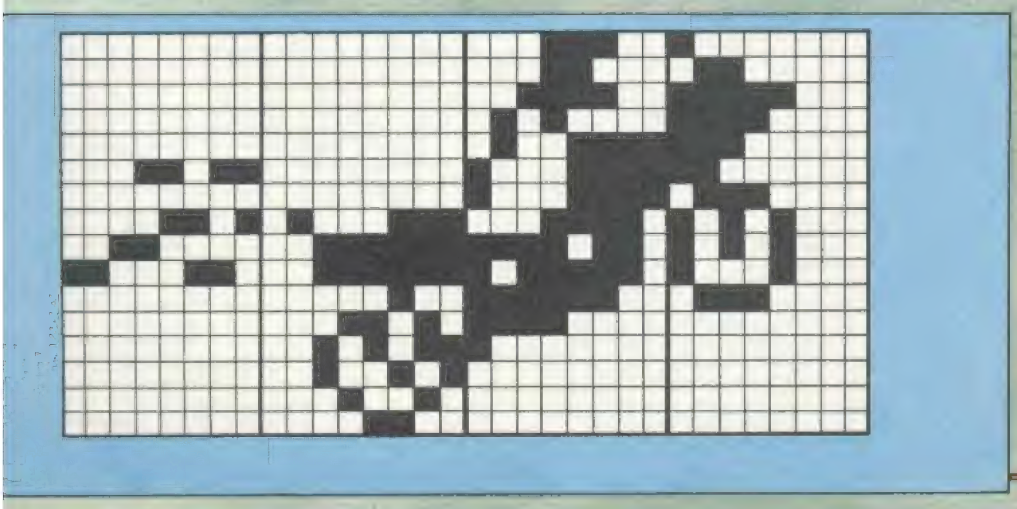


Neste artigo, os usuários do Apple e do ZX-81 verão como produzir figuras móveis. Usando os programas e observando as instruções, poderão, em seguida, criar seus próprios desenhos.

- EDITOR DE FIGURAS
- MOTO E SUBMARINO NO APPLE
- COMO MOVIMENTAR OS DESENHOS
- UM MONSTRO NO ZX-81

Para facilitar o trabalho, o programa a seguir cria uma "tabela de figuras" na memória do micro, a partir de padrões desenhados com asteriscos dentro de linhas **DATA**.

```
100  HOME :E = 35000: HIMEM: E:
 DIM A(100),B(100)
110  HTAB 10: VTAB 12: PRINT "U
M INSTANTE, POR FAVOR"
120  FOR I = 35000 TO 37000: PO
KE I,0: NEXT : HOME
130 F =  INT (E / 256)
140  POKE 232,E - F * 256: POKE
 233,F
150  PRINT "(DEFINICAO DA TABEL
A) LINHA DATA ";: PRINT "20 ,0
";: FOR II = 0 TO 19:W =  INT (
(42 + II * 32) / 256):O = 42 +
II * 32 - W * 256: PRINT ",";O;
" ,";W;" ";: NEXT II: PRINT : P
RINT : PRINT "RETURN P/ CONTINU
AR": INPUT "";O$: HOME
160  POKE E,20: POKE E + 1,0: F
OR II = 0 TO 19
170 W =  INT ((42 + II * 32) /
256):O = 42 + II * 32 - W * 256
: POKE E + 2 + 2 * II,O: POKE E
 + 3 + 2 * II,W
180  FOR I = 0 TO 100:B(I) = 0:
 NEXT I
190 Q = 0: GR : COLOR= 1: FOR I
 = 1 TO 8: READ Z$: FOR J = 1 T
O 8
200  IF  MID$ (Z$,J,1) = "*" TH
EN  PLOT J + 15,I + 29
210  NEXT J: NEXT I
220  HTAB 17: VTAB 21: PRINT "1
2345678"
230  GOTO 500
240  FOR V = 0 TO K - 1
250  IF B = 2 AND A(V) > 0 AND
A(V) < 4 THEN 280
260  IF B < 2 AND (A(V) > 0 OR
A(V) > 4) THEN 280
270 B = 0:Q = Q + 1
280 B(Q) = B(Q) + A(V) * (8 ^ B
)
290 B = B + 1
300  IF B > 2 THEN B = 0:Q = Q
+ 1
310  NEXT V
320  TEXT : HOME : PRINT "(FIGU
RA ";II + 1;") LINHA DATA  ";
330  FOR V = 0 TO 31
340  IF V <  > 0 THEN  PRINT ",
";
350  PRINT B(V);" ";
360  POKE E + 42 + 32 * II + V,
B(V)
370  NEXT V
380  PRINT : PRINT : PRINT "RET
URN P/ CONTINUAR": INPUT "";O$
390  HOME : HGR : SCALE= 1: ROT
= 0
400  FOR I = 150 TO 75 STEP  -
5
410  HCOLOR= 3
420  DRAW II + 1 AT 130,I
430  HCOLOR= 0
440  DRAW II + 1 AT 130,I
450  NEXT I
460  HCOLOR= 3
470  DRAW II + 1 AT 130,75
480  FOR I = 1 TO 50: NEXT I
490  NEXT II: END
500 X = 16:Y = 30:K = 1: FOR J
= 1 TO 4
510  FOR I = 1 TO 8
520 M = 1: IF  SCRN( X,Y) = 1 T
HEN M = M + 4
530 X = X + 1:A(K) = M:K = K +
1
540  NEXT I
550 M = 2: IF  SCRN( X,Y) = 1 T
HEN M = M + 4
560 Y = Y + 1:A(K) = M:K = K +
1
570  FOR I = 1 TO 8
580 M = 3: IF  SCRN( X,Y) = 1 T
HEN M = M + 4
590 X = X - 1:A(K) = M:K = K +
1
600  NEXT I
610 M = 2: IF  SCRN( X,Y) = 1 T
HEN M = M + 4
620 Y = Y + 1:A(K) = M:K = K +
1
630  NEXT J
640  GOTO 240
1000  REM    12345678
1001  DATA  "        "
1002  DATA  "        "
1003  DATA  "        "
1004  DATA  "        "
1005  DATA  "        "
1006  DATA  "        "
1007  DATA  "   **   "
1008  DATA  "      **"
```



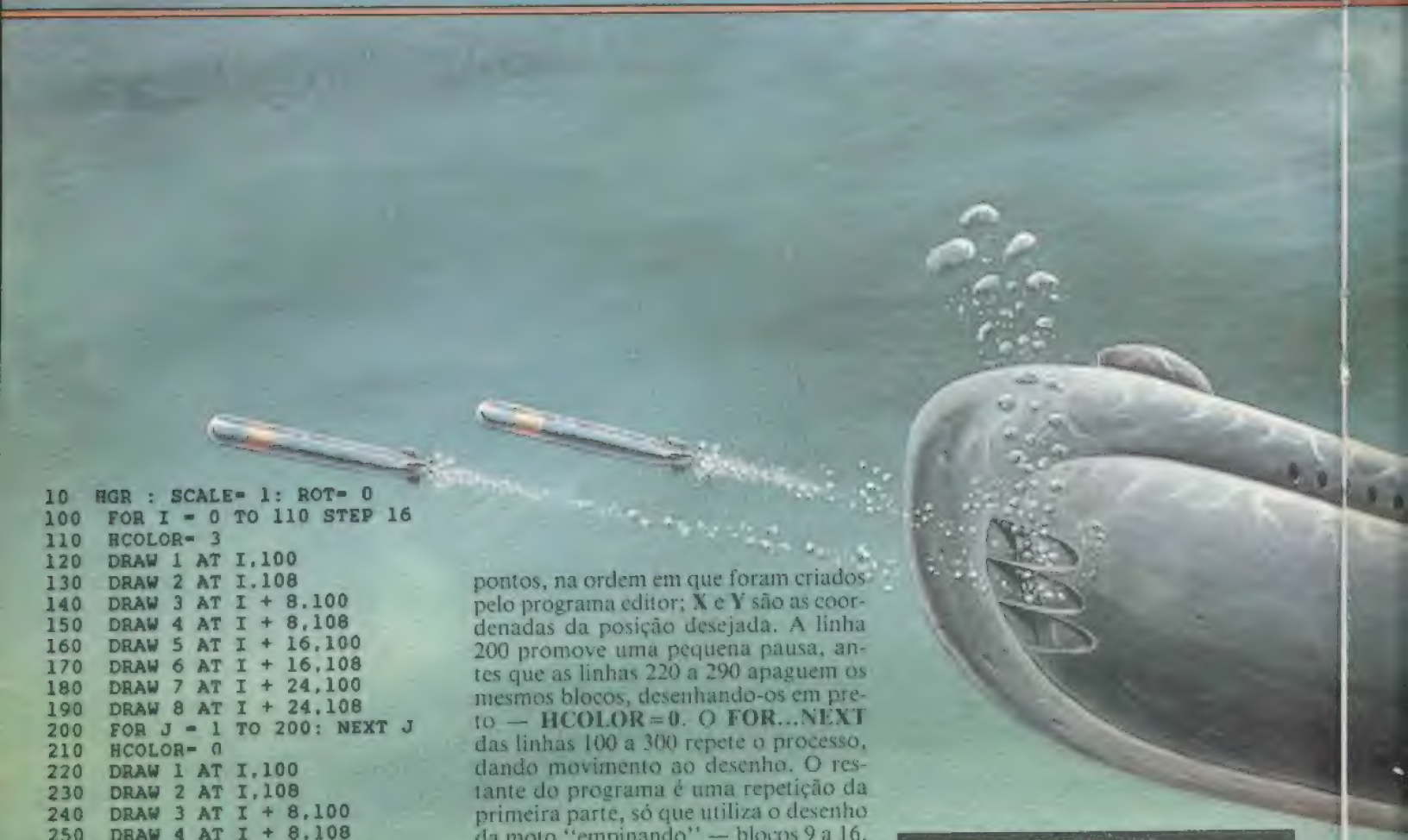
O programa permite que você crie figuras móveis, tomando como modelo os desenhos que aqui apresentamos. Esses desenhos são destinados a outros micros, onde as figuras compõem-se de blocos de oito por oito pontos.

Para utilizar o programa, adicione ao seu final linhas **DATA** correspondentes às figuras da moto das páginas 316 e 317. Cada linha **DATA** deve conter o padrão de uma das linhas de um bloco de oito por oito pontos, desenhado com asteriscos. Na listagem encontram-se as oito primeiras linhas **DATA**, correspondentes ao primeiro bloco da moto. Digite os blocos na seguinte ordem: bloco superior esquerdo, bloco inferior esquerdo, bloco superior da segunda coluna, e assim por diante, até o bloco inferior direito. Podem ser digitados até vinte blocos, ou 160 linhas **DATA**.

Para montar na memória do Apple uma tabela de figuras, digite **RUN**. O mesmo comando imprimirá as linhas **DATA** necessárias para criar a tabela de dentro de um programa **BASIC**. Os desenhos dos blocos digitados serão executados em baixa e alta resolução.

COMO UTILIZAR A TABELA DE FIGURAS

Tendo a tabela com os blocos dos desenhos da moto na memória — são dezesseis blocos ou 128 linhas **DATA** —, você poderá dar movimento a eles. Digite **NEW** e copie o programa:



```
10  HGR : SCALE= 1: ROT= 0
100  FOR I = 0 TO 110 STEP 16
110  HCOLOR= 3
120  DRAW 1 AT I,100
130  DRAW 2 AT I,108
140  DRAW 3 AT I + 8,100
150  DRAW 4 AT I + 8,108
160  DRAW 5 AT I + 16,100
170  DRAW 6 AT I + 16,108
180  DRAW 7 AT I + 24,100
190  DRAW 8 AT I + 24,108
200  FOR J = 1 TO 200: NEXT J
210  HCOLOR= 0
220  DRAW 1 AT I,100
230  DRAW 2 AT I,108
240  DRAW 3 AT I + 8,100
250  DRAW 4 AT I + 8,108
260  DRAW 5 AT I + 16,100
270  DRAW 6 AT I + 16,108
280  DRAW 7 AT I + 24,100
290  DRAW 8 AT I + 24,108
300  NEXT I
310  FOR I = 110 TO 250 STEP 16
320  HCOLOR= 3
330  DRAW 9 AT I,100
340  DRAW 10 AT I,108
350  DRAW 11 AT I + 8,100
360  DRAW 12 AT I + 8,108
370  DRAW 13 AT I + 16,100
380  DRAW 14 AT I + 16,108
390  DRAW 15 AT I + 24,100
400  DRAW 16 AT I + 24,108
410  FOR J = 1 TO 100: NEXT J
420  HCOLOR= 0
430  DRAW 9 AT I,100
440  DRAW 10 AT I,108
450  DRAW 11 AT I + 8,100
460  DRAW 12 AT I + 8,108
470  DRAW 13 AT I + 16,100
480  DRAW 14 AT I + 16,108
490  DRAW 15 AT I + 24,100
500  DRAW 16 AT I + 24,108
510  NEXT I
```

Na linha 10 **HGR** liga a tela de alta resolução; **SCALE = 1** define a escala do desenho e **ROT = 0** define a orientação horizontal do mesmo.

As linhas 120 a 190 desenham a moto, usando **DRAW N AT X,Y**, onde **N** é o número do bloco de oito por oito pontos, na ordem em que foram criados pelo programa editor; **X** e **Y** são as coordenadas da posição desejada. A linha 200 promove uma pequena pausa, antes que as linhas 220 a 290 apaguem os mesmos blocos, desenhando-os em preto — **HCOLOR = 0**. O **FOR...NEXT** das linhas 100 a 300 repete o processo, dando movimento ao desenho. O restante do programa é uma repetição da primeira parte, só que utiliza o desenho da moto "empinando" — blocos 9 a 16.

Para montar o desenho sem empregar o programa editor, adicione ao programa as linhas **DATA** que ele criou. As linhas seguintes usarão os números ali contidos para montar a mesma tabela de figuras.

```
20  HOME :E = 35000: HIMEM: E
30 F =  INT (E / 256): POKE 232
,E - F * 256: POKE 233,F
40  FOR I = E TO E + 41 + 16 *
32
50  READ A: POKE I,A
60  NEXT
```

E contém o endereço inicial do local onde a tabela será montada na memória — é igual à variável de mesmo nome no programa editor. **HIMEM:E** proteje esta área no topo da memória. Os **POKE** da linha 20 informam o endereço ao Apple. O restante do programa lê os dados com **READ** e monta a tabela com **POKE**. Na linha 40, 16 é o número total de blocos.

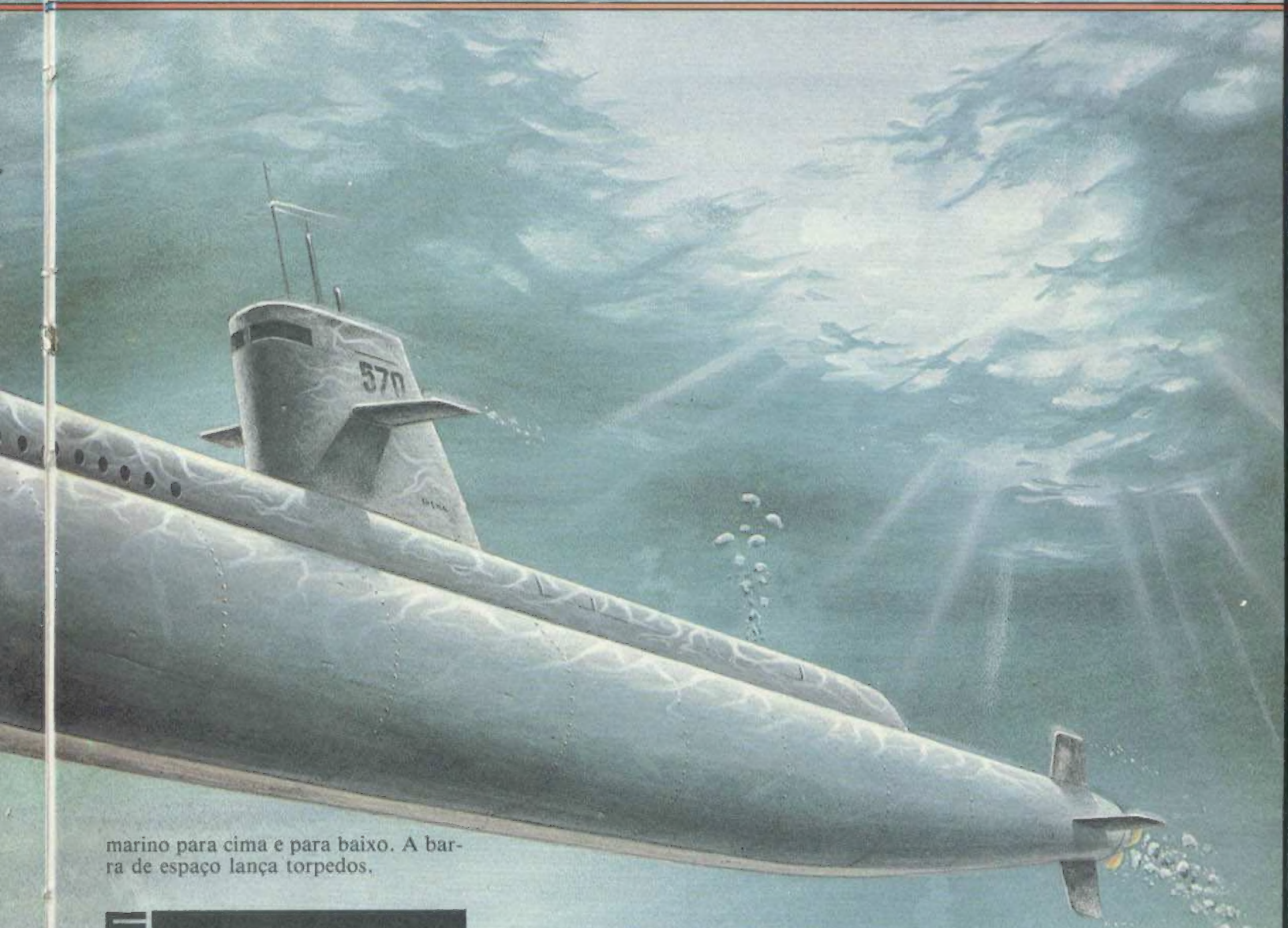
Outra alternativa para evitar o uso do editor é gravar a tabela de figuras em fita, usando o comando **W** do monitor, ou em disco, usando o comando **BSAVE**. As linhas **DATA** deixarão de ser necessárias, mas continuaremos precisando das linhas 20 a 40.

O SUBMARINO

O programa seguinte movimenta o submarino da página 319, desde que ele tenha sido criado pelo programa editor.

```
10  HGR : SCALE= 1: ROT= 0
100 Y = 100:LY = Y: GOTO 170
110  GET A$: IF A$ = "" THEN 11
0
120 LY = Y
130  IF A$ = "P" AND Y > 3 THEN
 Y = Y - 3: GOTO 170
140  IF A$ = "L" AND Y < 150 TH
EN Y = Y + 3: GOTO 170
150  IF A$ = " " THEN 260
160  GOTO 110
170  HCOLOR= 0
180  DRAW 1 AT 10,LY
190  DRAW 2 AT 18,LY
200  DRAW 3 AT 26,LY
210  HCOLOR= 5
220  DRAW 1 AT 10,Y
230  DRAW 2 AT 18,Y
240  DRAW 3 AT 26,Y
250  GOTO 110
260  FOR I = 0 TO 230 STEP 4
270  HCOLOR= 1
280  DRAW 4 AT 34 + I,Y
290  HCOLOR= 0
300  DRAW 4 AT 34 + I,Y
310  NEXT I: GOTO 110
```

As teclas R e L movimentam o sub-



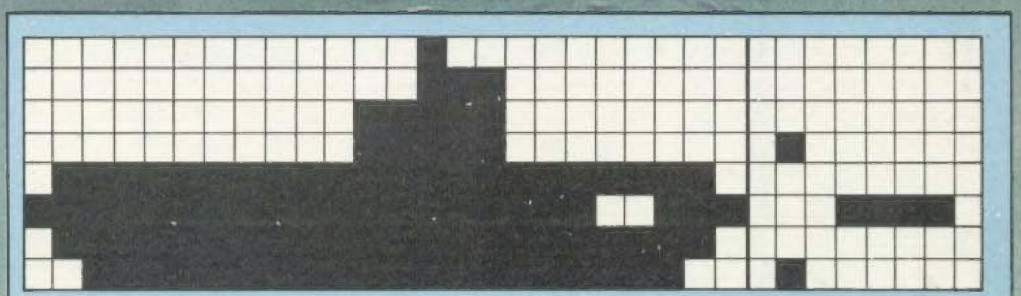
marino para cima e para baixo. A barra de espaço lança torpedos.

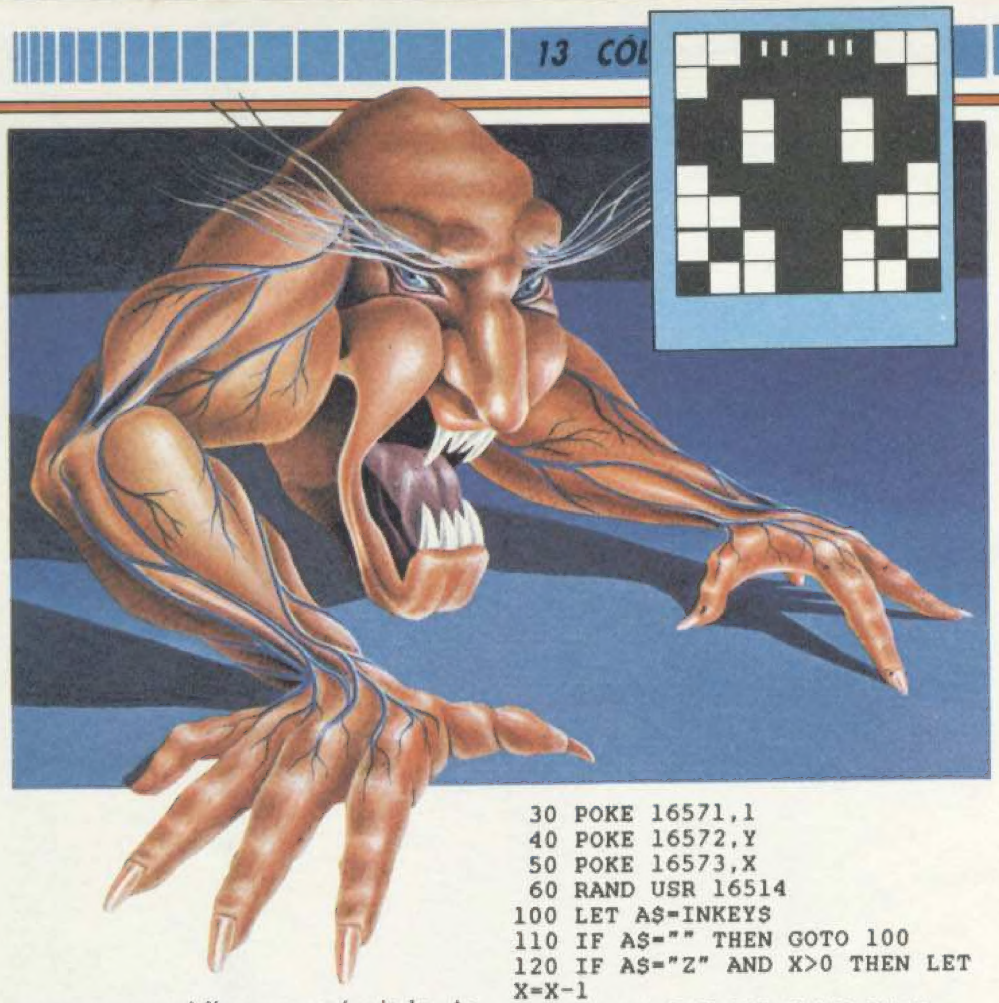
Monstro

O programa que se segue cria um monstro com auxílio do código de máquina.

```
 1 REM ...................
10 LET A$="2A0C40233ABC4047112
100FE00280319"
15 LET A$=A$+"10FD3ABD4016005F
1911BE403ABB40FE"
20 LET A$=A$+"00280311CE400604
C506041A77231310"
25 LET A$=A$+"FA011D0009C110F0
C9010000"
30 LET A$=A$+"0000000000000000
0000000000000000"
35 LET A$=A$+"878B8B0480850580
0280800106850586"
50 FOR N=16514 TO 16605
60 POKE N,16*CODE A$+CODE A$(2
)-476
70 LET A$=A$(3 TO )
80 NEXT N
```

A linha 1 precisa ter pelo menos 92 caracteres após **REM**, para acomodar o





programa em código que será criado. As linhas 10 a 25 contêm o programa que coloca o monstro na tela — os códigos estão dentro de **A$**. Os zeros — 32 ao todo — da linha 30 apagam o desenho com espaços em branco.

A linha 35 contém o padrão do monstro. Se compararmos seu conteúdo — dois dígitos de cada vez — com o conjunto de caracteres do apêndice A do manual, veremos que ali estão caracteres gráficos invertidos.

Esses caracteres desenham o monstro, do canto superior esquerdo ao inferior direito. Como eles dividem um espaço de caractere em quatro partes, a resolução final é de oito por oito pontos.

Para mudar o desenho, basta trocar os números nesta linha. Códigos de comandos BASIC que tenham mais de um caractere com **AT**, **TAB**, **THEN** ou **LEN** não podem ser empregados.

Quando utilizamos **RUN**, o programa contido em **A$** é colocado no espaço que se segue a **REM** pelas linhas 50 a 80. Se listarmos, então, o programa, veremos os caracteres correspondentes aos códigos. Os que desenham o monstro estão no final da linha.

Depois que o programa estiver dentro da linha **REM**, apague as demais e digite:

```
10 LET X=14
20 LET Y=8
30 POKE 16571,1
40 POKE 16572,Y
50 POKE 16573,X
60 RAND USR 16514
100 LET A$=INKEY$
110 IF A$="" THEN GOTO 100
120 IF A$="Z" AND X>0 THEN LET
X=X-1
130 IF A$="X" AND X<28 THEN
LET X=X+1
140 IF A$="P" AND Y>0 THEN LET
Y=Y-1
150 IF A$="L" AND Y<20 THEN LET
Y=Y+1
160 POKE 16571,0
170 RAND USR 16514
180 GOTO 30
```

As linhas 10 e 20 colocam as coordenadas do centro da tela em **X** e **Y**. A linha 30 introduz o número 1 no programa em código, usando **POKE**, para que seja impresso o monstro, e não espaços em branco. As linhas 40 e 50 colocam **X** e **Y** da mesma forma, estabelecendo a posição da figura. A seguir, a rotina em código é chamada e o computador desenha o monstro.

As linhas 100 a 150 contêm a rotina típica que move algo na tela (veja página 31). **Z** movimenta o desenho para a esquerda, **X** para a direita, **P** para cima e **L** para baixo.

A linha 160 coloca 0 dentro do programa em código, para desenhar espaços em branco; a linha 170 chama o programa, apagando o desenho. A linha 180 volta ao início.

INVERSÃO DA TELA

Com o código de máquina, pode-se também inverter a tela no ZX-81. O programa a seguir faz isso, trocando o preto pelo branco e vice-versa. Embora ele deva ser chamado de dentro de um programa BASIC, coloque-o na memória usando o nosso monitor (veja página 92).

```
2A 0C 40 06 17 23 7E FE 76 28
05 C6 80 77 18 F5 10 F3 C9
```

Use **RAND USR "endereço inicial"** para rodá-lo. Como esse comando, sem um número na frente, limpará a tela, o efeito não será interessante. Um programa simples dará melhores resultados.

```
20 LIST
30 RAND USR
```

Aqui, **RAND USR** deve ser seguido do endereço inicial do programa.

Podemos ainda colocar essa rotina de inversão dentro do programa do monstro. Os códigos serão adicionados a **A$**. Acrescente a linha:

```
40 LET A$=A$+"2A0C400617237EFE7
62805C6807718F510F3C9"
```

Note que não há espaço entre os códigos.

A linha 50 deve ser alterada para colocar os códigos extras dentro do **REM**.

```
50 FOR N=16514 TO 16624
```

A linha **REM** precisa ter mais 19 espaços disponíveis, pelo menos. Após rodar o programa com essas modificações, apague todas as linhas, com exceção da **REM**, e copie o programa de movimentação com esta linha a mais:

```
155 IF A$="I" THEN RAND USR 166
06
```

O endereço 16606 é o início da nova rotina. Ela poderá ser chamada por meio da tecla I. Ao pressioná-la, o monstro será invertido. Mantendo a pressão, ele cintilará.

SIMULE ALTA RESOLUÇÃO

Como você já sabe, o ZX-81 não tem alta resolução gráfica. Porém, com o código de máquina, pode-se produzir algo semelhante. Este programa simplesmente usa **POKE** para colocar um pequeno programa após o **REM** da linha 10 e, depois, o chama.

```
10 REM .............
20 POKE 16514,62
30 POKE 16516,237
40 POKE 16517,71
50 POKE 16518,201
60 FOR N=0 TO 30
70 POKE 16515,N
80 RAND USR 16514
90 NEXT N
```

Infelizmente, não há um processo fácil para mover desenhos desse tipo.

CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR LINHA

LINHA	FABRICANTE	MODELO
Apple II +	Appletronica	Thor 2010
Apple II +	CCE	MC-4000 Exato
Apple II +	CPA	Absolutus
Apple II +	CPA	Polaris
Apple II +	Digitus	DGT-AP
Apple II +	Dismac	D-8100
Apple II +	ENIAC	ENIAC II
Apple II +	Franklin	Franklin
Apple II +	Houston	Houston AP
Apple II +	Magnex	DM II
Apple II +	Maxitronica	MX-2001
Apple II +	Maxitronica	MX-48
Apple II +	Maxitronica	MX-64
Apple II +	Maxitronica	Maxitronic I
Apple II +	Microcraft	Craf II Plus
Apple II +	Milmar	Apple II Plus
Apple II +	Milmar	Apple Master
Apple II +	Milmar	Apple Senior
Apple II +	Omega	MC-400
Apple II +	Polymax	Maxxi
Apple II +	Polymax	Poly Plus
Apple II +	Spectrum	Microengenho I
Apple II +	Spectrum	Spectrum ed
Apple II +	Suporte	Venus II
Apple II +	Sycomig	SIC I
Apple II +	Unitron	AP II
Apple II +	Victor do Brasil	Elppa II Plus
Apple II +	Victor do Brasil	Elppa Jr.
Apple IIe	Microcraft	Craft IIe
Apple IIe	Microdigital	TK-3000 IIe
Apple IIe	Spectrum	Microengenho II
MSX	Gradiente	Expert GPC-1
MSX	Sharp	Hotbit HB-8000
Sinclair Spectrum	Microdigital	TK-90X
Sinclair Spectrum	Timex	Timex 2000
Sinclair ZX-81	Apply	Apply 300
Sinclair ZX-81	Engebras	AS-1000
Sinclair ZX-81	Filcres	NEZ-8000
Sinclair ZX-81	Microdigital	TK-82C
Sinclair ZX-81	Microdigital	TK-83
Sinclair ZX-81	Microdigital	TK-85
Sinclair ZX-81	Prologica	CP-200
Sinclair ZX-81	Ritas	Ringo R-470
Sinclair ZX-81	Timex	Timex 1000
Sinclair ZX-81	Timex	Timex 1500
TRS-80 Mod. I	Dismac	D-8000
TRS-80 Mod. I	Dismac	D-8001/2
TRS-80 Mod. I	LNW	LNW-80
TRS-80 Mod. I	Video Genie	Video Genie I
TRS-80 Mod.III	Digitus	DGT-100
TRS-80 Mod.III	Digitus	DGT-1000
TRS-80 Mod.III	Kemitron	Naja 800
TRS-80 Mod.III	Prologica	CP-300
TRS-80 Mod.III	Prologica	CP-500
TRS-80 Mod.III	Sysdata	Sysdata III
TRS-80 Mod.III	Sysdata	Sysdata Jr.
TRS-80 Mod.IV	Multix	MX-Compacto
TRS-80 Mod.IV	Sysdata	Sysdata IV
TRS-Color	Codimex	CS-6508
TRS-Color	Dynacom	MX-1600
TRS-Color	LZ	Color 64
TRS-Color	Microdigital	TKS-800
TRS-Color	Prologica	CP-400

CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR NOME

FABRICANTE	MODELO	PAÍS	LINHA
Appletronica	Thor 2010	Brasil	Apple II +
Apply	Apply 300	Brasil	Sinclair ZX-81
CCE	MC-4000 Exato	Brasil	Apple II +
CPA	Absolutus	Brasil	Apple II +
CPA	Polaris	Brasil	Apple II +
Codimex	CS-6508	Brasil	TRS-Color
Digitus	DGT-100	Brasil	TRS-80 Mod.III
Digitus	DGT-1000	Brasil	TRS-80 Mod.III
Digitus	DGT-AP	Brasil	Apple II +
Dismac	D-8000	Brasil	TRS-80 Mod. I
Dismac	D-8001/2	Brasil	TRS-80 Mod. I
Dismac	D-8100	Brasil	Apple II +
Dynacom	MX-1600	Brasil	TRS-Color
ENIAC	ENIAC II	Brasil	Apple II +
Engebras	AS-1000	Brasil	Sinclair ZX-81
Filcres	NEZ-8000	Brasil	Sinclair ZX-81
Franklin	Franklin	USA	Apple II +
Gradiente	Expert GPC1	Brasil	MSX
Houston	Houston AP	Brasil	Apple II +
Kemitron	Naja 800	Brasil	TRS-80 Mod.III
LNW	LNW-80	USA	TRS-80 Mod. I
LZ	Color 64	Brasil	TRS-Color
Magnex	DM II	Brasil	Apple II +
Maxitronica	MX-2001	Brasil	Apple II +
Maxitronica	MX-48	Brasil	Apple II +
Maxitronica	MX-64	Brasil	Apple II +
Maxitronica	Maxitronic I	Brasil	Apple II +
Microcraft	Craft II Plus	Brasil	Apple II +
Microcraft	Caft IIe	Brasil	Apple IIe
Microdigital	TK-3000 IIe	Brasil	Apple IIe
Microdigital	TK-82C	Brasil	Sinclair ZX-81
Microdigital	TK-83	Brasil	Sinclair ZX-81
Microdigital	TK-85	Brasil	Sinclair ZX-81
Microdigital	TK-90X	Brasil	Sinclair Spectrum
Microdigital	TKS-800	Brasil	TRS-Color
Milmar	Apple II Plus	Brasil	Apple II +
Milmar	Apple Master	Brasil	Apple II +
Milmar	Apple Senior	Brasil	Apple II +
Multix	MX-Compacto	Brasil	TRS-80 Mod.IV
Omega	MC-400	Brasil	Apple II +
Polymax	Maxxi	Brasil	Apple II +
Polymax	Poly Plus	Brasil	Apple II +
Prologica	CP-200	Brasil	Sinclair ZX-81
Prologica	CP-300	Brasil	TRS-80 Mod.III
Prologica	CP-400	Brasil	TRS-Color
Prologica	CP-500	Brasil	TRS-80 Mod.III
Ritas	Ringo R-470	Brasil	Sinclair ZX-81
Sharp	Hotbit HB-8000	Brasil	MSX
Spectrum	Microengenho I	Brasil	Apple II +
Spectrum	Microengenho II	Brasil	Apple IIe
Spectrum	Spectrum ed	Brasil	Apple II +
Suporte	Venus II	Brasil	Apple II +
Sycomig	SIC I	Brasil	Apple II +
Sysdata	Sysdata III	Brasil	TRS-80 Mod.III
Sysdata	Sysdata IV	Brasil	TRS-80 Mod.IV
Sysdata	Sysdata Jr.	Brasil	TRS-80 Mod.III
Timex	Timex 1000	USA	Sinclair ZX-81
Timex	Timex 1500	USA	Sinclair ZX-81
Timex	Timex 2000	USA	Sinclair Spectrum
Unitron	AP II	Brasil	Apple II +
Victor do Brasil	Elppa II Plus	Brasil	Apple II +
Victor do Brasil	Elppa Jr.	Brasil	Apple II +
Video Genie	Video Genie I	USA	TRS-80 Mod. I

UM LOGOTIPO PARA CADA MODELO DE COMPUTADOR

INPUT foi especialmente projetado para microcomputadores compatíveis com as sete principais linhas existentes no mercado.
Os blocos de textos e listagens de programas aplicados apenas a determinadas linhas de micros podem ser identificados por meio dos seguintes símbolos:

Sinclair ZX-81

TRS-80

TK-2000

MSX

Spectrum

TRS-Color

Apple II

Quando o emblema for seguido de uma faixa, então tanto o texto como os programas que se seguem passam a ser específicos para a linha indicada.

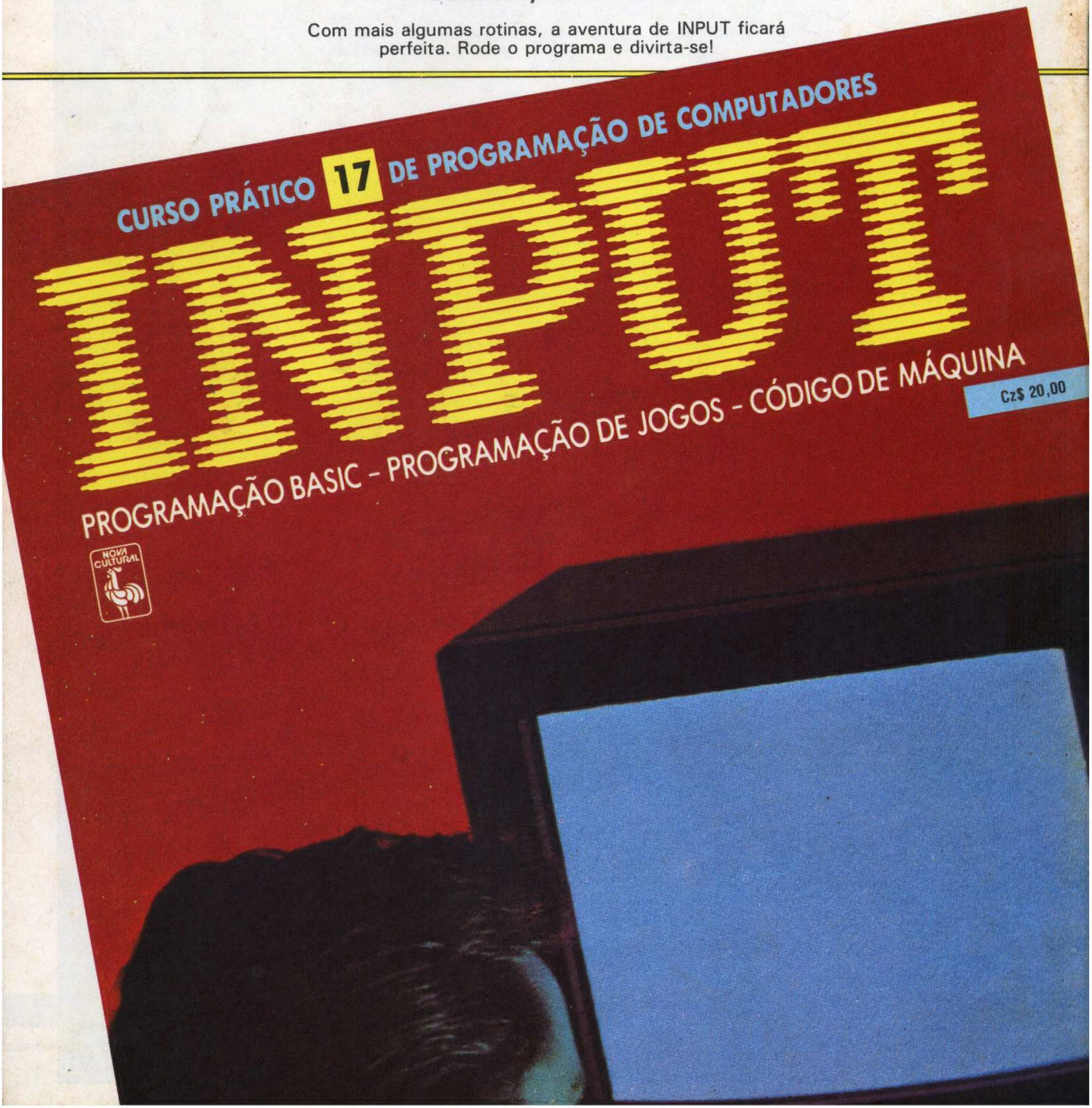
NO PRÓXIMO NÚMERO

APLICAÇÕES

Para aperfeiçoar sua datilografia, complete o programa do curso e pratique com textos mais longos.

PROGRAMAÇÃO BASIC

As funções matemáticas permitem controlar várias operações no micro. Recorra a elas para sofisticar seus gráficos.

PROGRAMAÇÃO DE JOGOS

Com mais algumas rotinas, a aventura de INPUT ficará perfeita. Rode o programa e divirta-se!